DupHunter: Flexible High-Performance Deduplication for Docker Registries

Nannan Zhao, Hadeel Albahar, Subil Abraham, Keren Chen, Vasily Tarasov, Dimitrios Skourtis, Lukas Rupprecht, Ali Anwar, and Ali R. Butt

Containers are ubiquitous

OS

Database
ORACLE DATABASE
MySQL

Web server
NGINX

Cache
redis

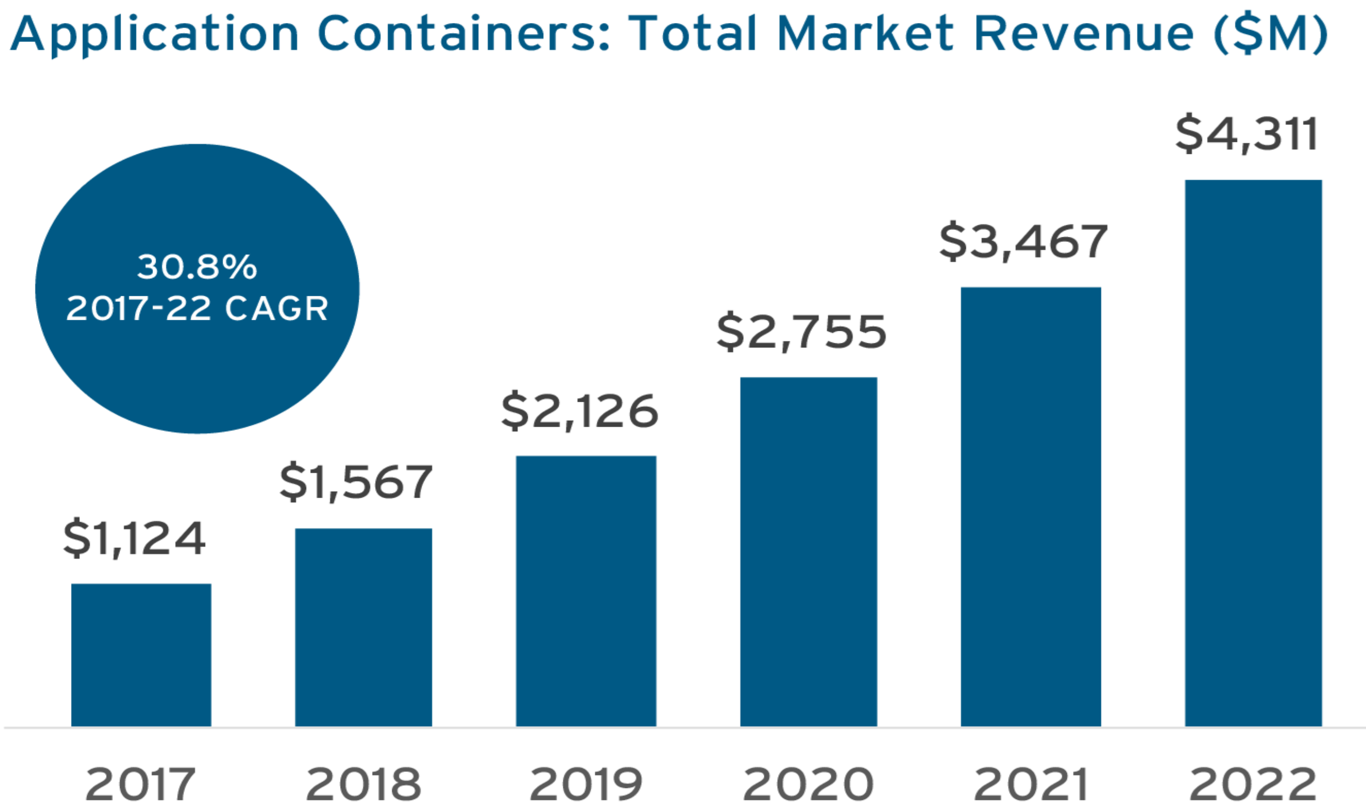
Serverless
APACHE OpenWhisk™

Deep learning
TensorFlow
mxnet

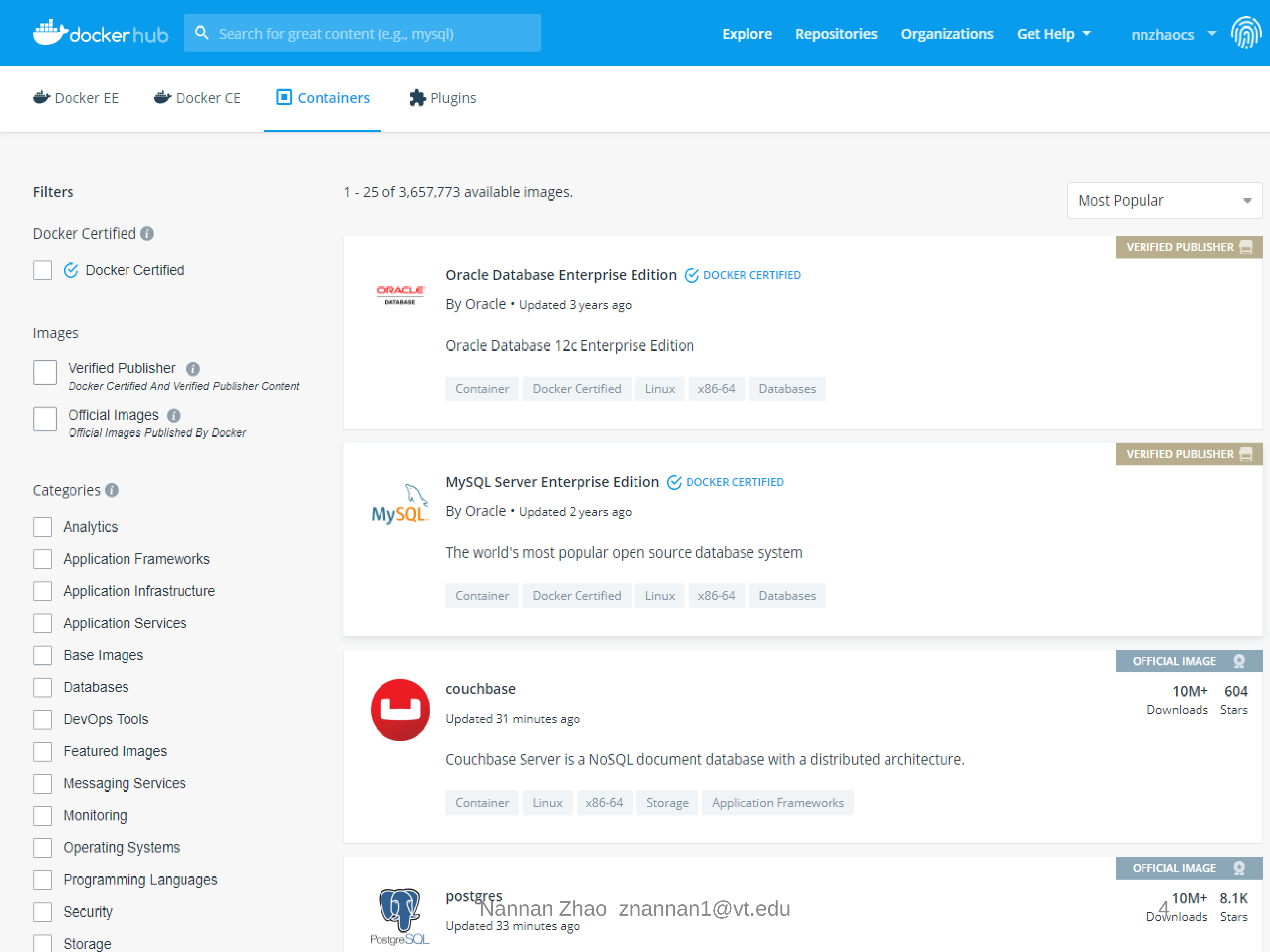
Big data
Spark
STORM

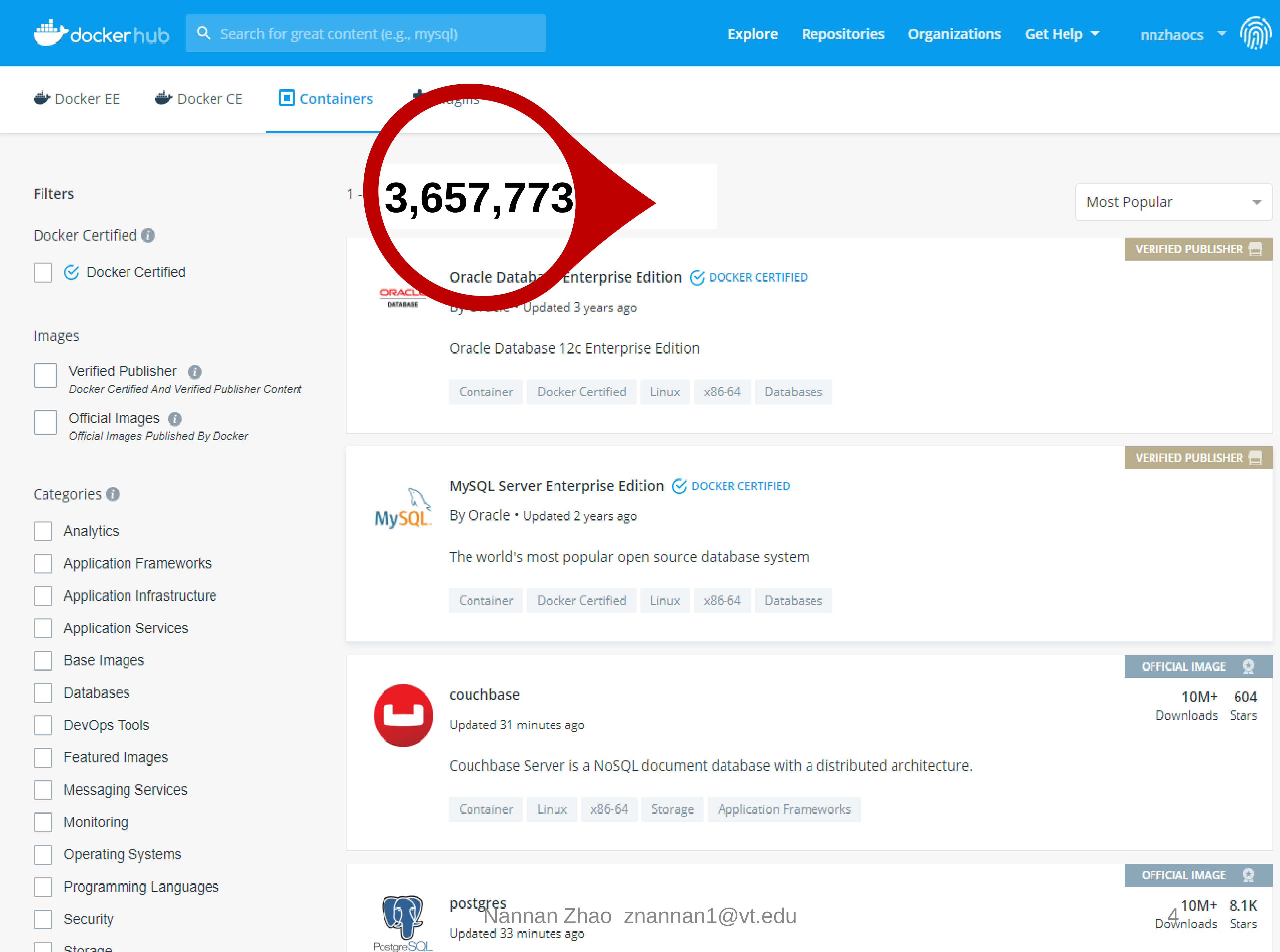
Languages
Ruby

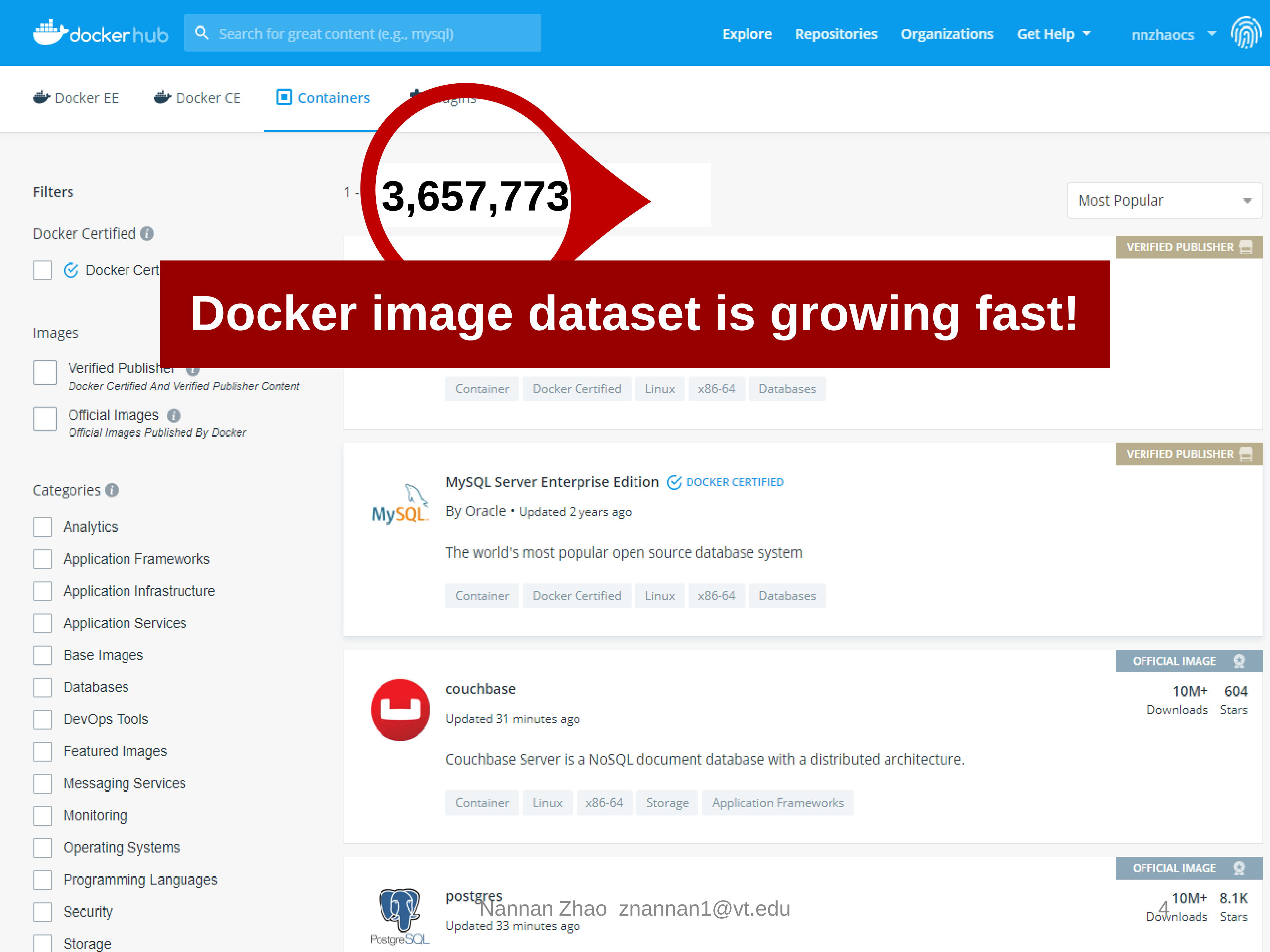
Application containerization is becoming a significant market player

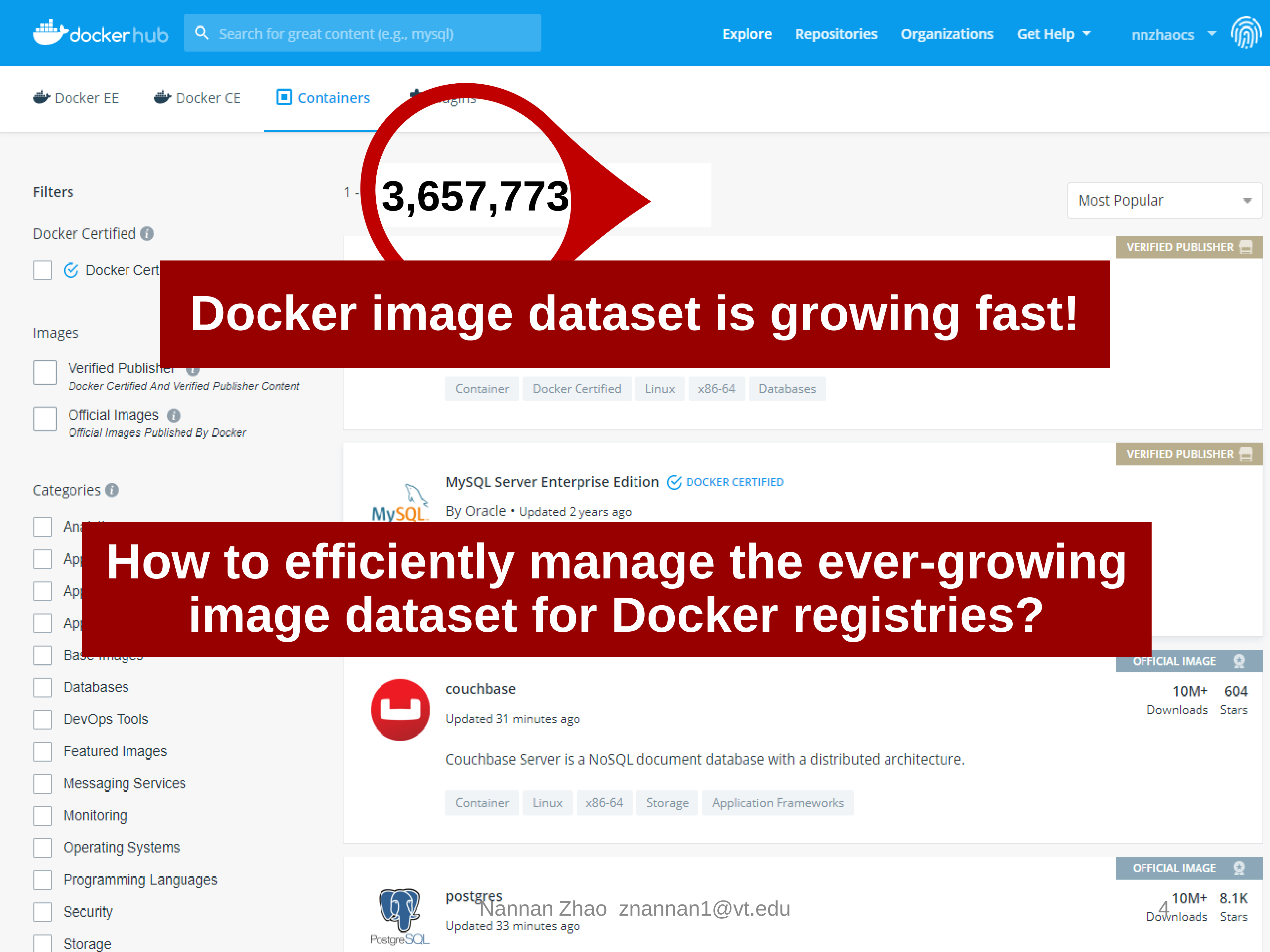


Source: 451 Research's Market Monitor: Cloud-Enabling Technologies - Application Containers, November 2018


docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organizations
Get Help
nnzhaocs
Docker EE
Docker CE
Containers
Plugins
Filters
Docker Certified
Docker Certified
Images
Verified Publisher
Docker Certified And Verified Publisher Content
Official Images
Official Images Published By Docker
Categories
Analytics
Application Frameworks
Application Infrastructure
Application Services
Base Images
Databases
DevOps Tools
Featured Images
Messaging Services
Monitoring
Operating Systems
Programming Languages
Security
Storage
1 - 25 of 3,657,773 available images.
Most Popular
VERIFIED PUBLISHER
Oracle Database Enterprise Edition
DOCKER CERTIFIED
By Oracle • Updated 3 years ago
Oracle Database 12c Enterprise Edition
Container
Docker Certified
Linux
x86-64
Databases
VERIFIED PUBLISHER
MySQL Server Enterprise Edition
DOCKER CERTIFIED
By Oracle • Updated 2 years ago
The world's most popular open source database system
Container
Docker Certified
Linux
x86-64
Databases
OFFICIAL IMAGE
couchbase
Updated 31 minutes ago
Couchbase Server is a NoSQL document database with a distributed architecture.
Container
Linux
x86-64
Storage
Application Frameworks
10M+ Downloads
604 Stars
OFFICIAL IMAGE
postgres
Updated 33 minutes ago
10M+ Downloads
8.1K Stars


docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organizations
Get Help
nnzhaocs
Docker EE
Docker CE
Containers
3,657,773
Most Popular
Filters
Docker Certified
Docker Certified
Images
Verified Publisher
Docker Certified And Verified Publisher Content
Official Images
Official Images Published By Docker
Categories
Analytics
Application Frameworks
Application Infrastructure
Application Services
Base Images
Databases
DevOps Tools
Featured Images
Messaging Services
Monitoring
Operating Systems
Programming Languages
Security
Storage
VERIFIED PUBLISHER
Oracle Datab Enterprise Edition
DOCKER CERTIFIED
Updated 3 years ago
Oracle Database 12c Enterprise Edition
Container
Docker Certified
Linux
x86-64
Databases
VERIFIED PUBLISHER
MySQL Server Enterprise Edition
DOCKER CERTIFIED
By Oracle • Updated 2 years ago
The world's most popular open source database system
Container
Docker Certified
Linux
x86-64
Databases
OFFICIAL IMAGE
couchbase
Updated 31 minutes ago
10M+ Downloads
604 Stars
Couchbase Server is a NoSQL document database with a distributed architecture.
Container
Linux
x86-64
Storage
Application Frameworks
OFFICIAL IMAGE
postgres
Updated 33 minutes ago
10M+ Downloads
8.1K Stars


docker hub
Search for great content (e.g., mysql)
Explore
Repositories
Organizations
Get Help
nnzhaocs
Docker EE
Docker CE
Containers
3,657,773
Most Popular
Filters
Docker Certified
Images
Verified Publisher
Docker Certified And Verified Publisher Content
Official Images
Official Images Published By Docker
Categories
Analytics
Application Frameworks
Application Infrastructure
Application Services
Base Images
Databases
DevOps Tools
Featured Images
Messaging Services
Monitoring
Operating Systems
Programming Languages
Security
Storage
Docker image dataset is growing fast!
VERIFIED PUBLISHER
Container
Docker Certified
Linux
x86-64
Databases
MySQL Server Enterprise Edition
DOCKER CERTIFIED
By Oracle • Updated 2 years ago
The world's most popular open source database system
OFFICIAL IMAGE
couchbase
Updated 31 minutes ago
Couchbase Server is a NoSQL document database with a distributed architecture.
Storage
Application Frameworks
10M+
Downloads
604
Stars
postgres
Updated 33 minutes ago
8.1K



Nannan Zhao znannan1@vt.edu

Our contribution: DupHunter—a framework to deduplicate images in Docker registries

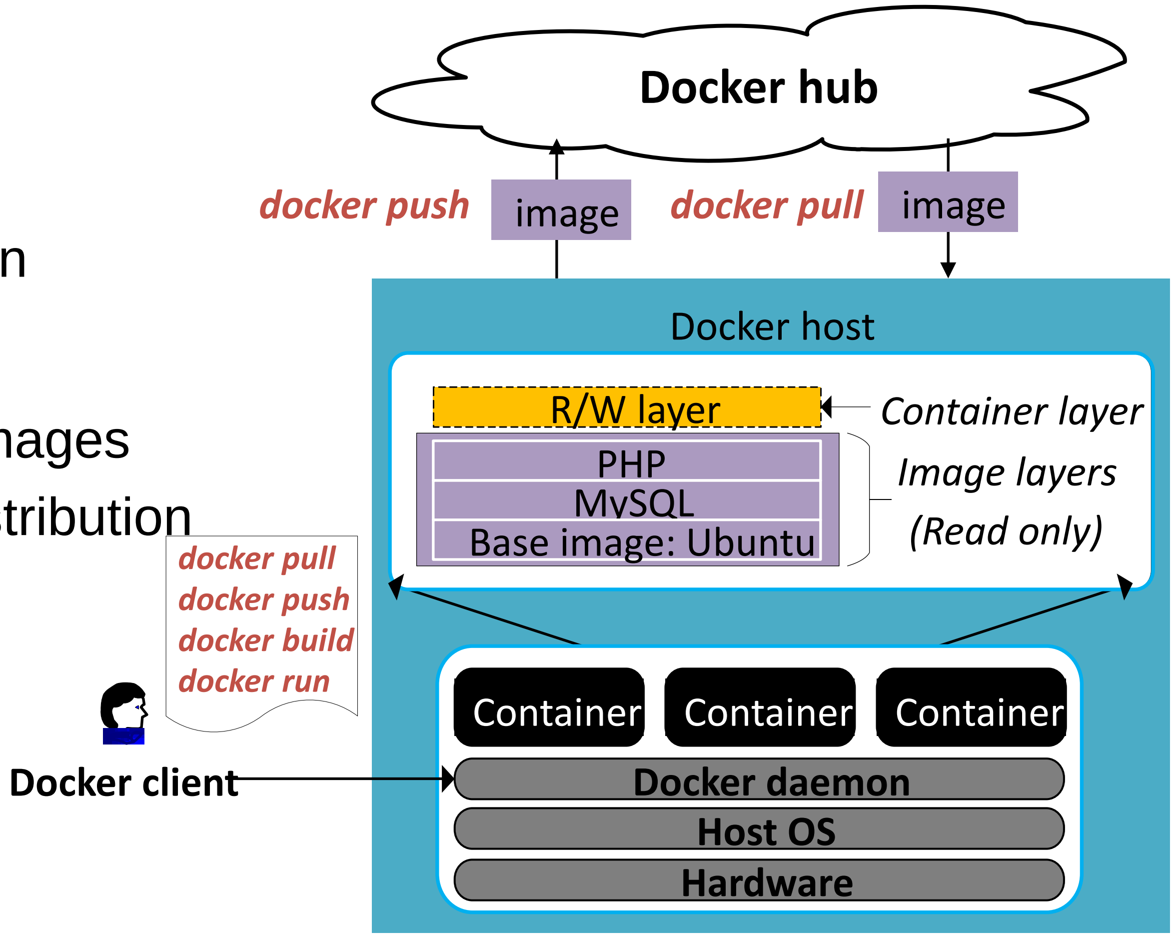
- We make two key observations:
  1. Container images exhibit a lot of redundancy.
  2. User access pattern is predictable.

- We design DupHunter to work with compressed images and provide layer deduplication and reduce layer restore overhead.

- We evaluate DupHunter with representative real world workloads. Compared to the state of the art, DupHunter:
  - reduces storage **space** by up to **6.9x.**
  - reduces the **GET layer latency** up to **2.8x.**

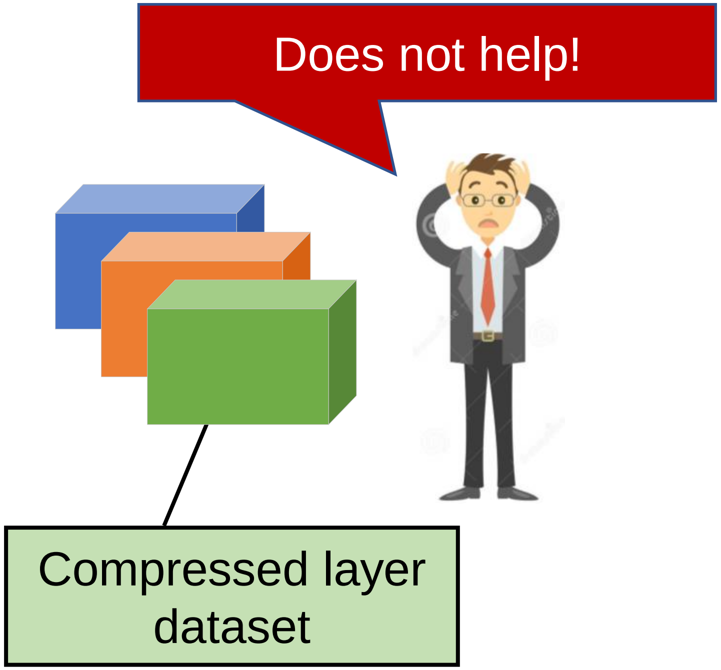
Overview of Docker

- ❑ Docker container is a self-contained executable package, that is:
  - Lightweight
  - Portable
  - Provides Isolation
- ❑ Docker registry:
  - Stores Docker images
  - Supports fast distribution
  - Facilitates easy deployment



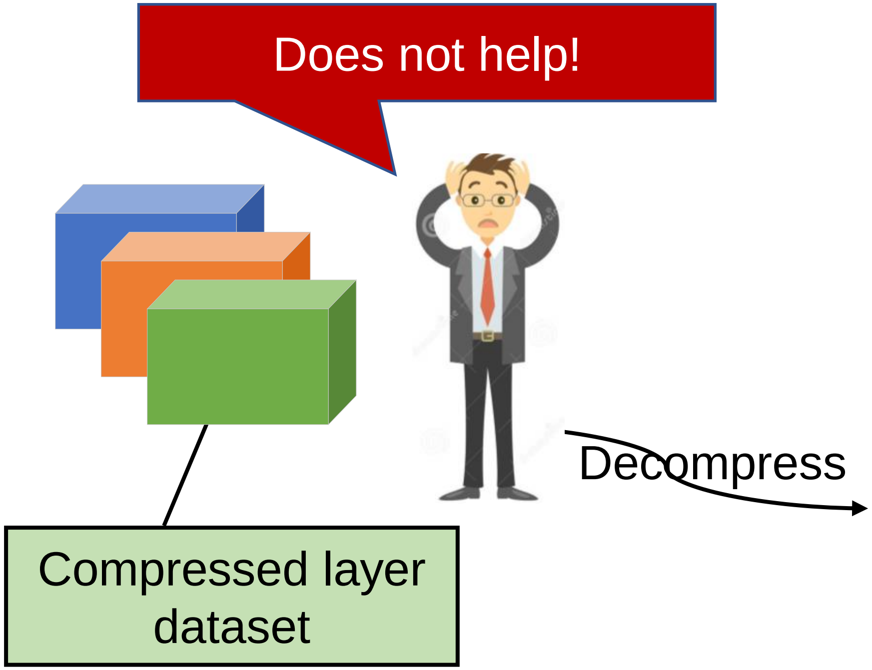
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.



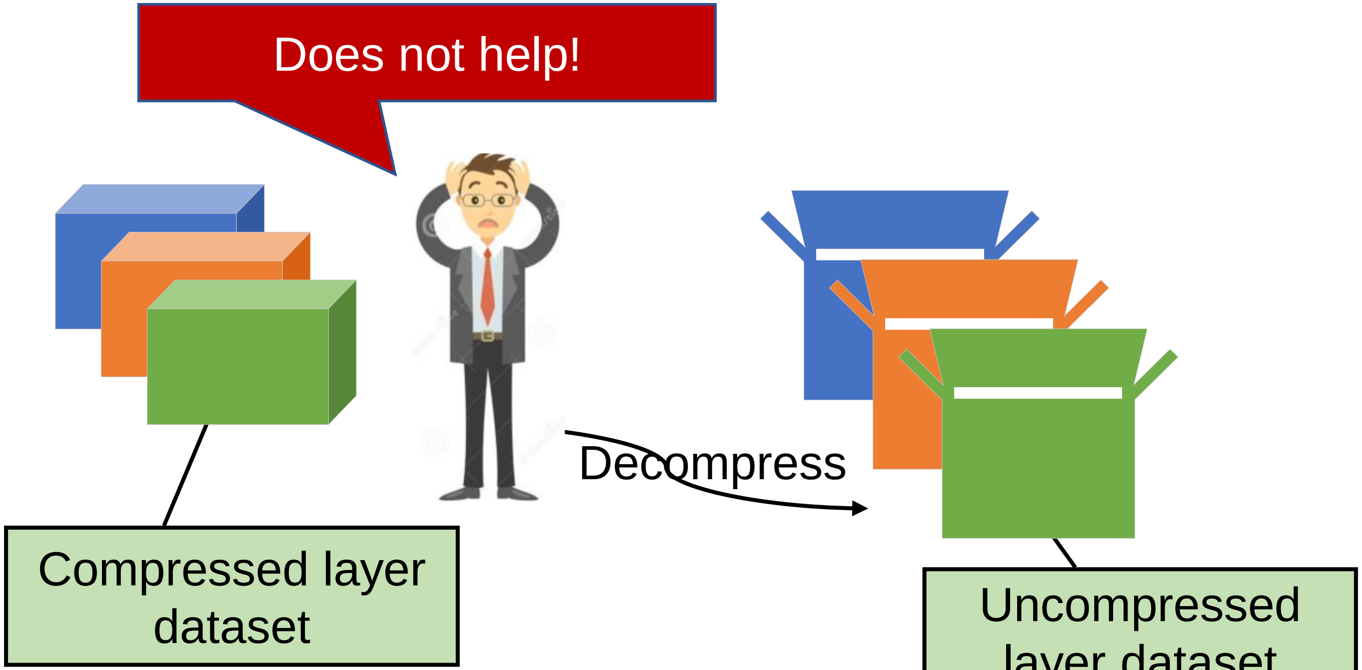
Key observation I: Image dataset has large amount of redundant files

- ❑ Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- ❑ Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.



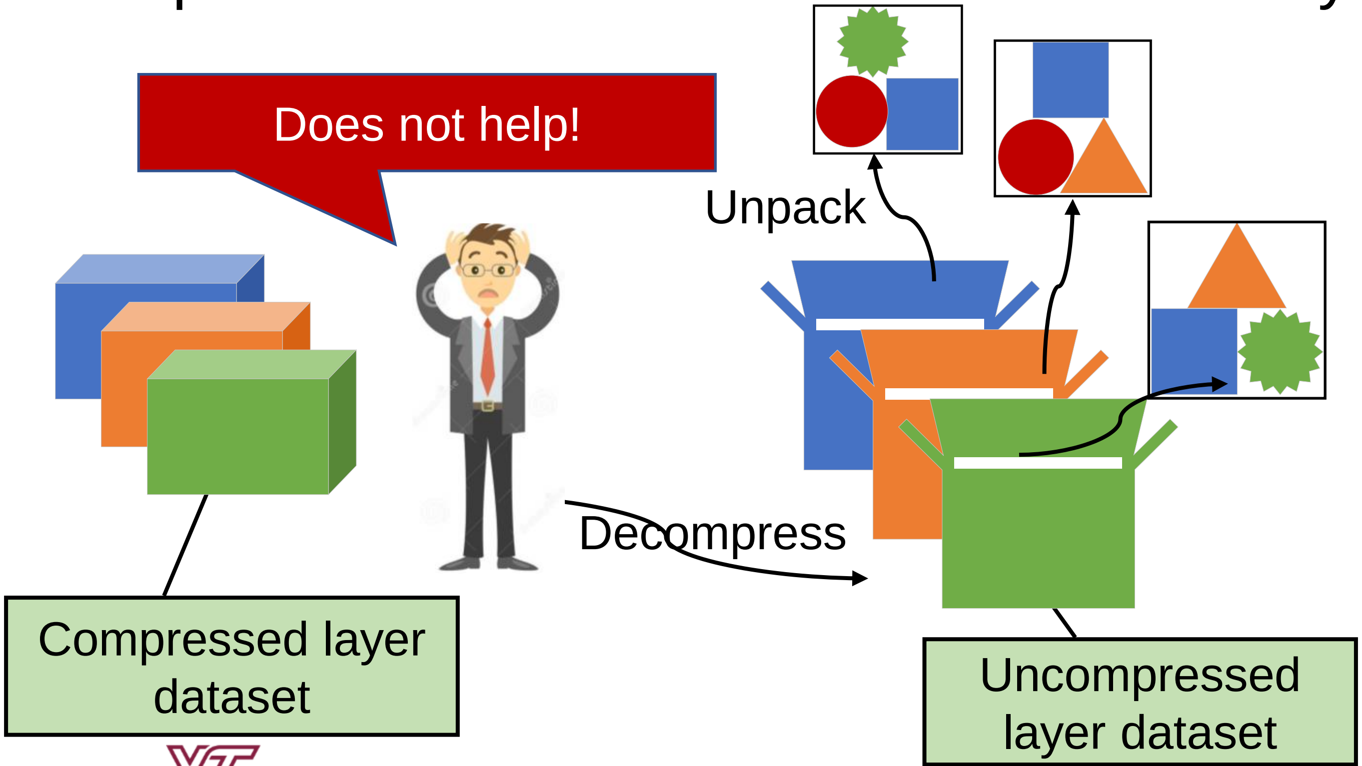
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.



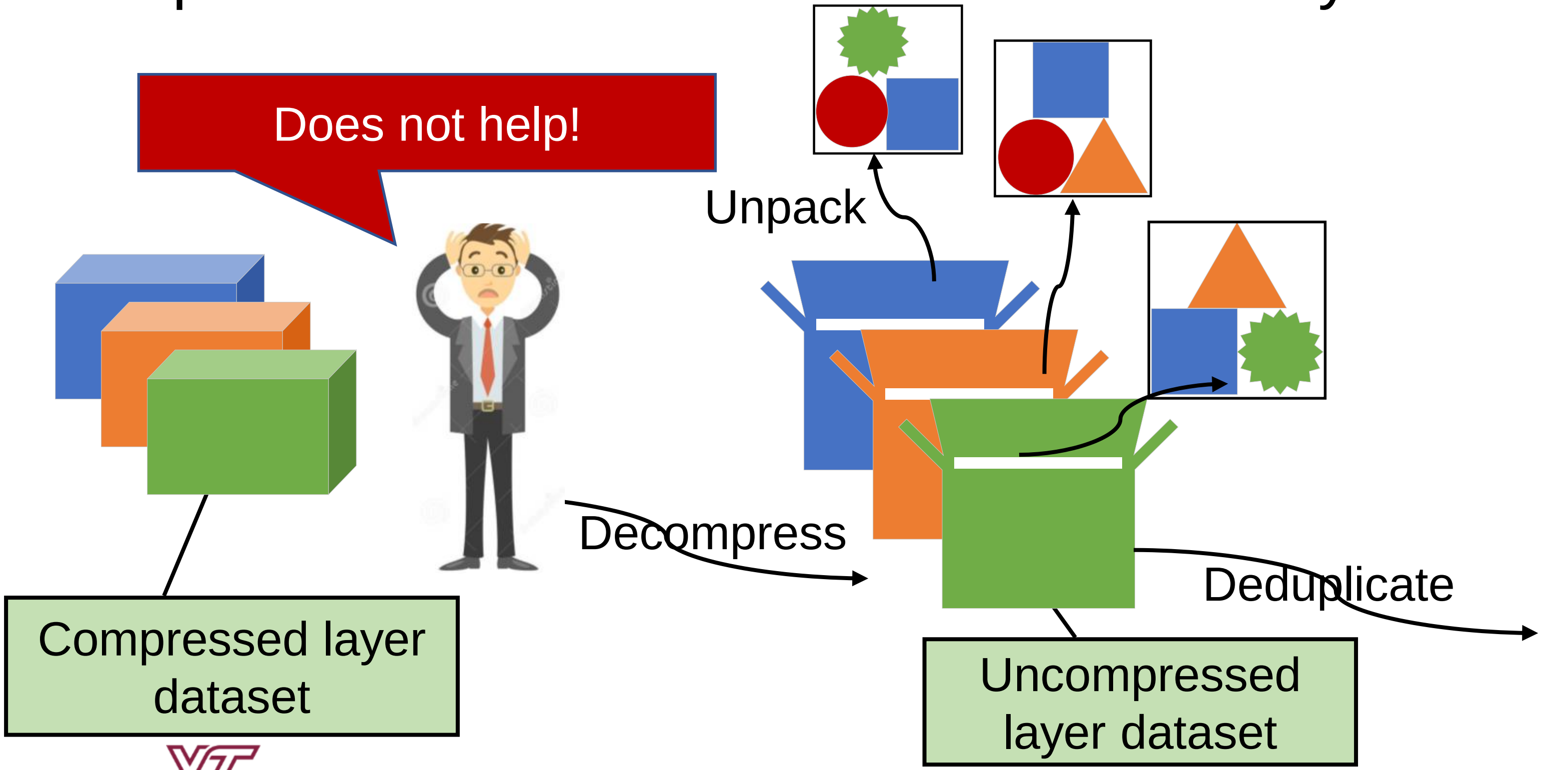
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.



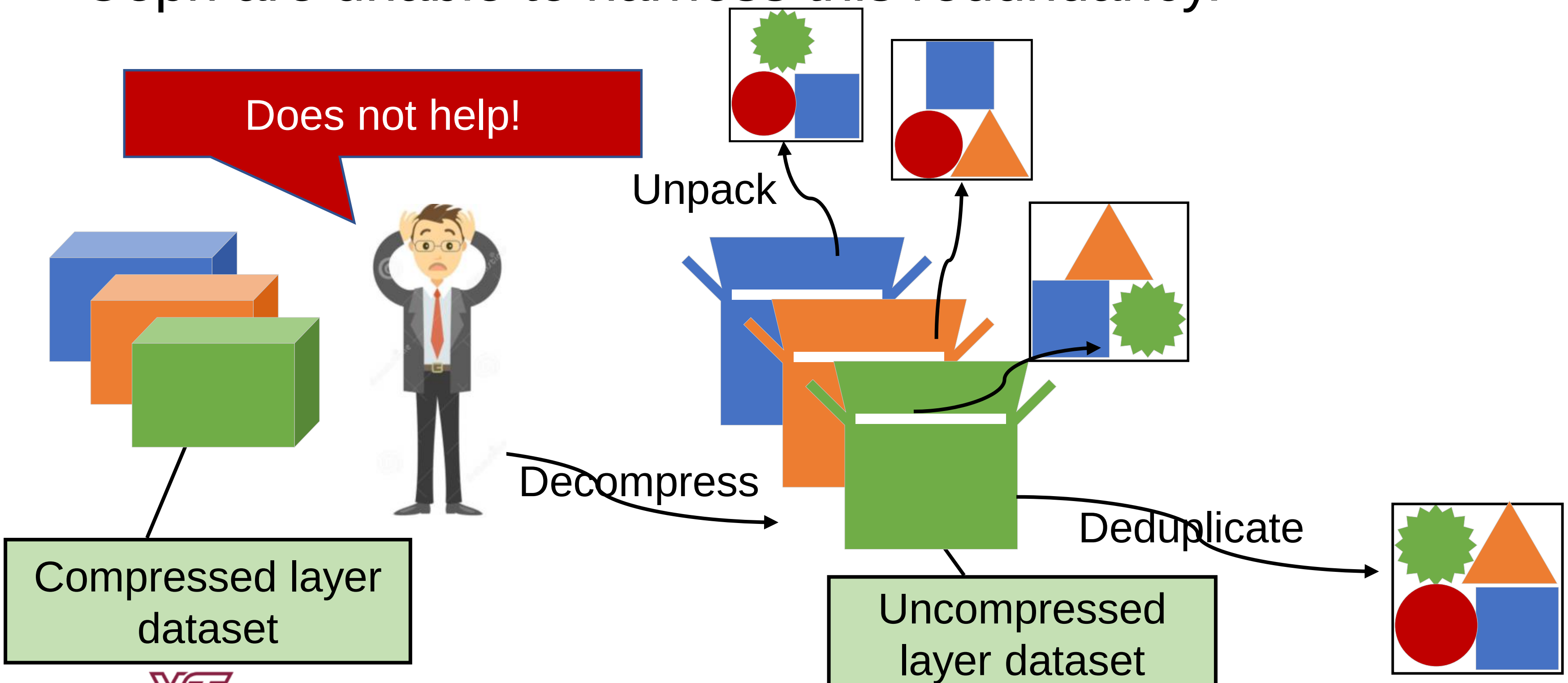
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.



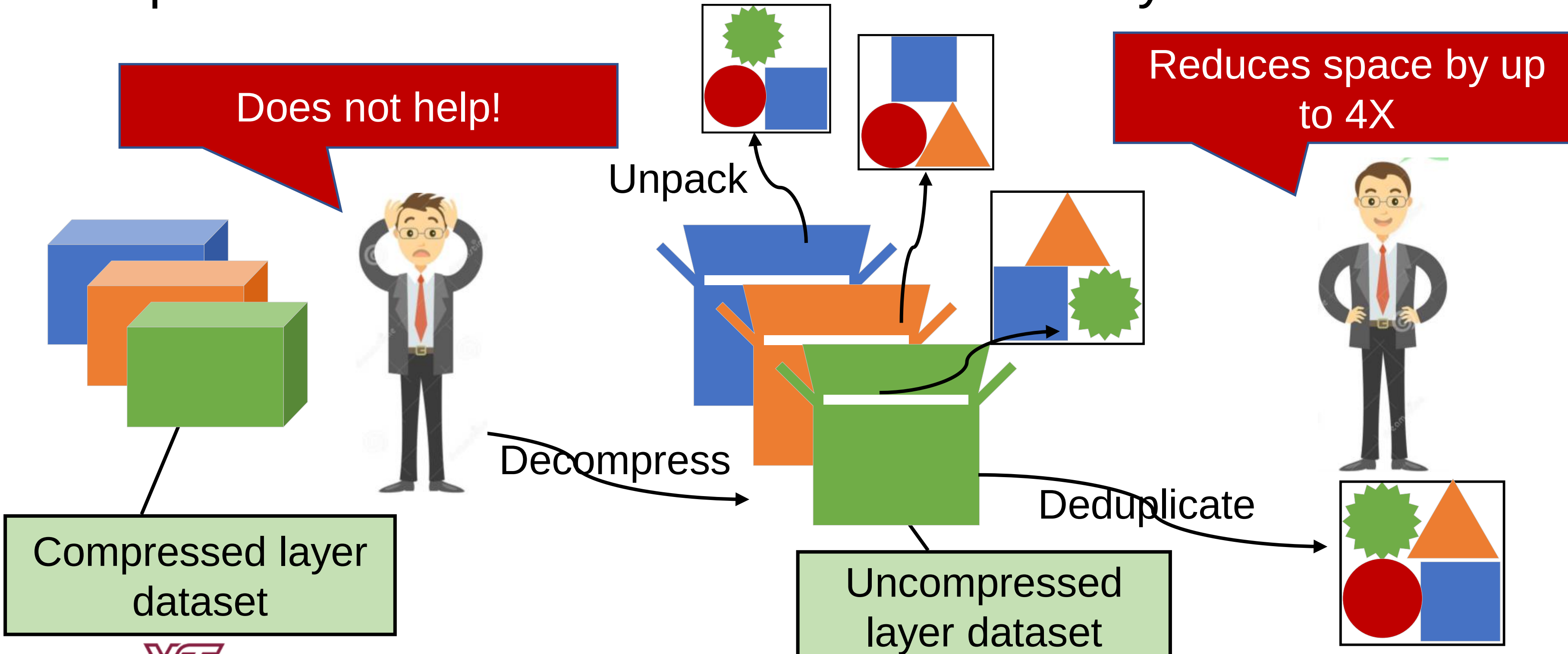
Key observation I: Image dataset has large amount of redundant files

- ❑ Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- ❑ Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.



Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.



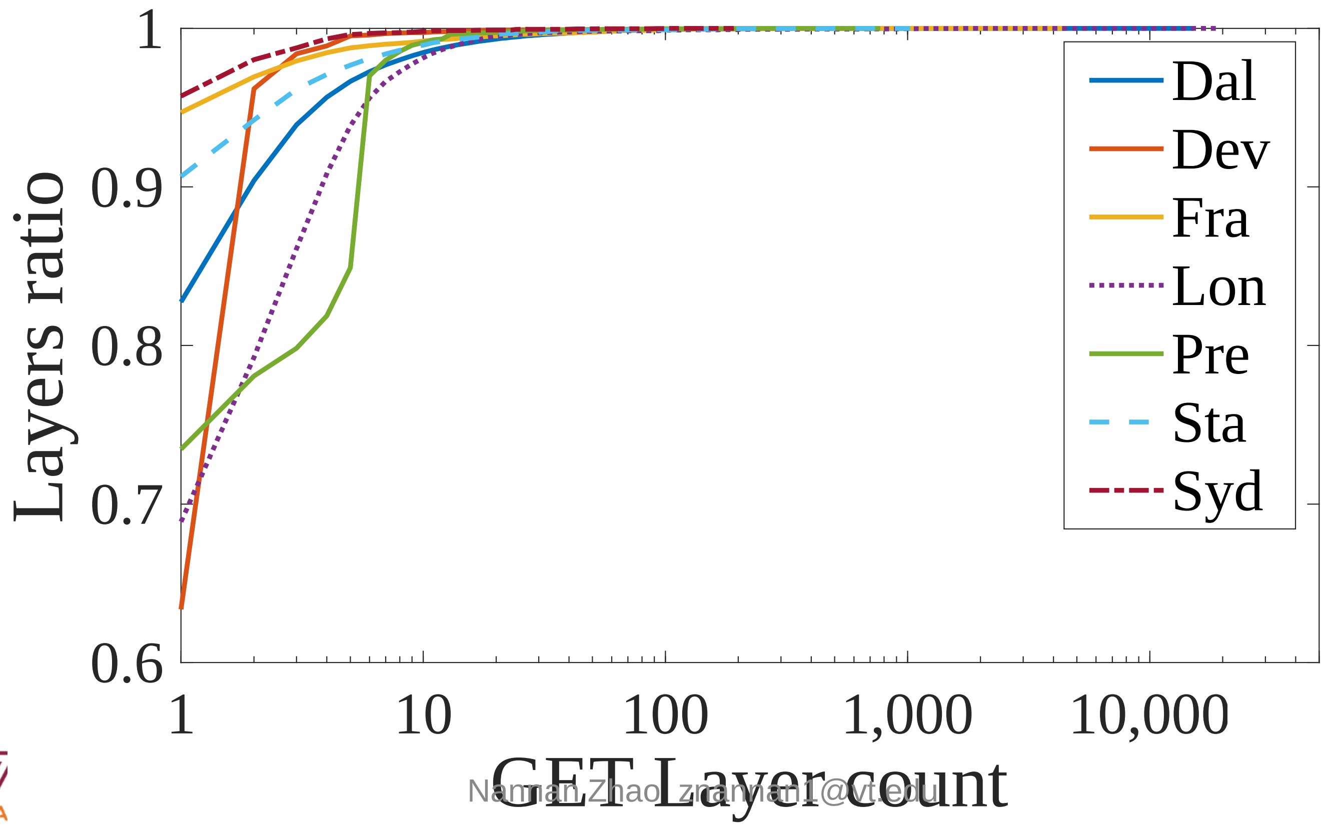
Key observation I: Image dataset has large amount of redundant files

- Container images have a lot of redundancy.
  - 97% of files across layers are duplicates!
- Existing technologies such as Jdupes, VDO, Btrfs, ZFS, and Ceph are unable to harness this redundancy.

Layer restore incurs considerable overhead for layer pulling latency up to 98x!

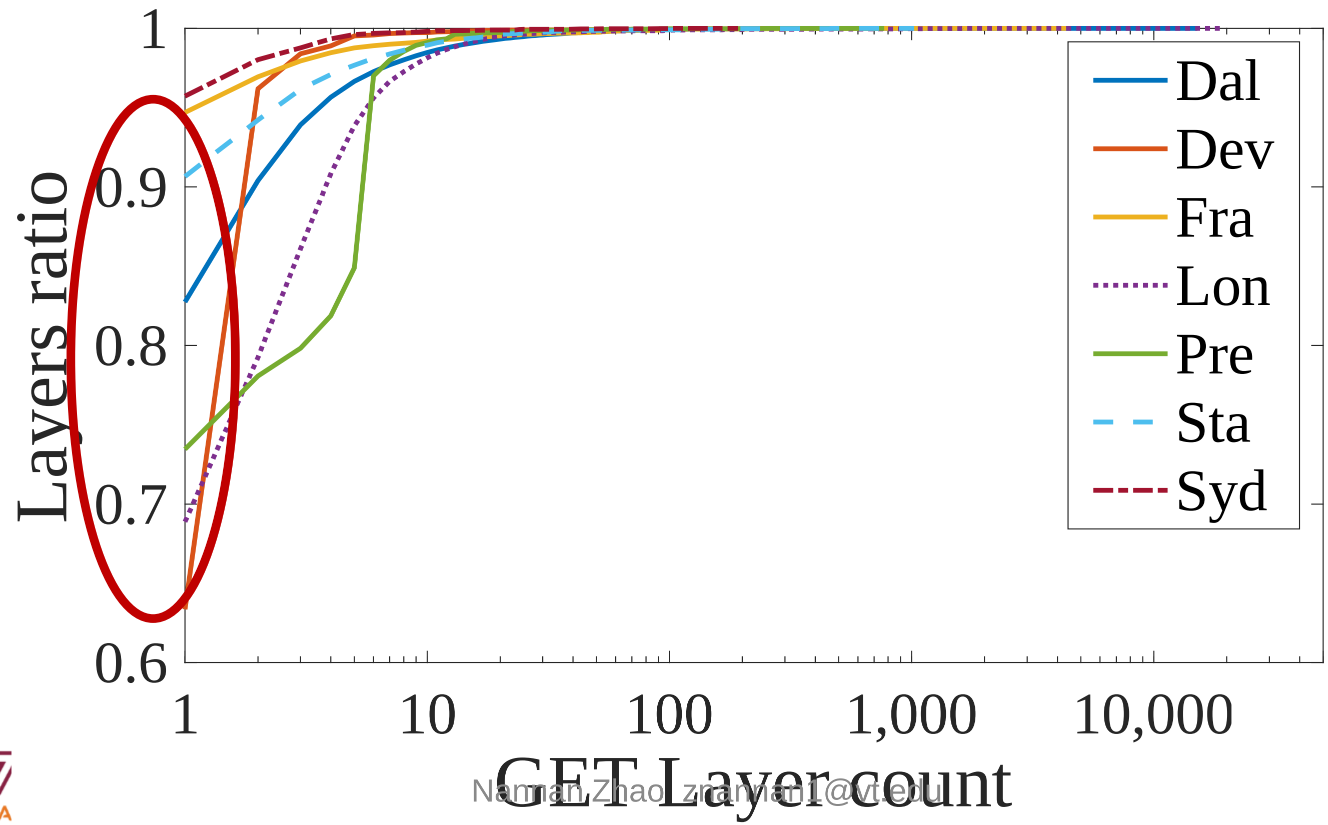
Key observation II: Predictable user access pattern

- We observe a consistent user pulling pattern: Pull manifest first, then layers, but not all of the layers will be pulled.
- We performed a quantitive study using a 75-day IBM Cloud Registry workload with 7 availability zones.



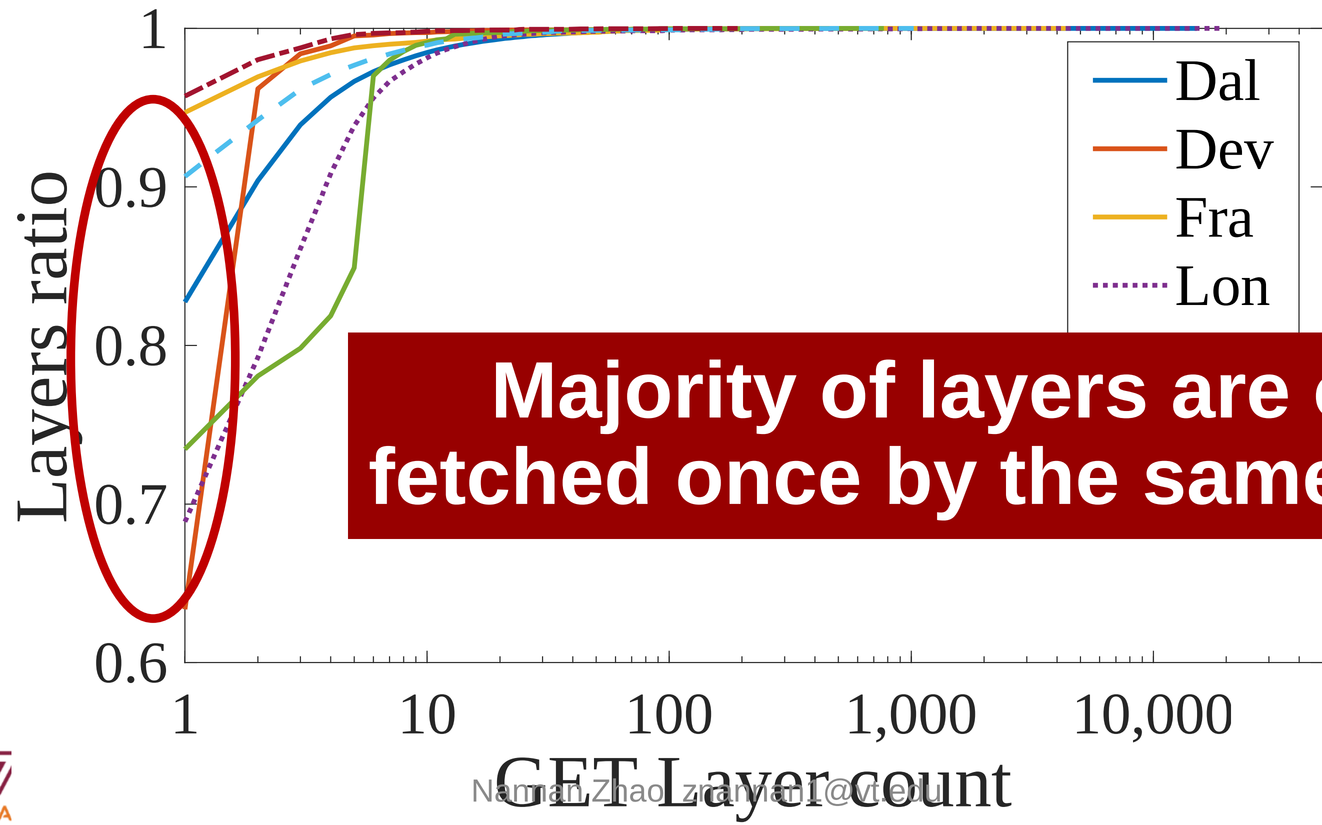
Key observation II: Predictable user access pattern

- We observe a consistent user pulling pattern: Pull manifest first, then layers, but not all of the layers will be pulled.
- We performed a quantitive study using a 75-day IBM Cloud Registry workload with 7 availability zones.



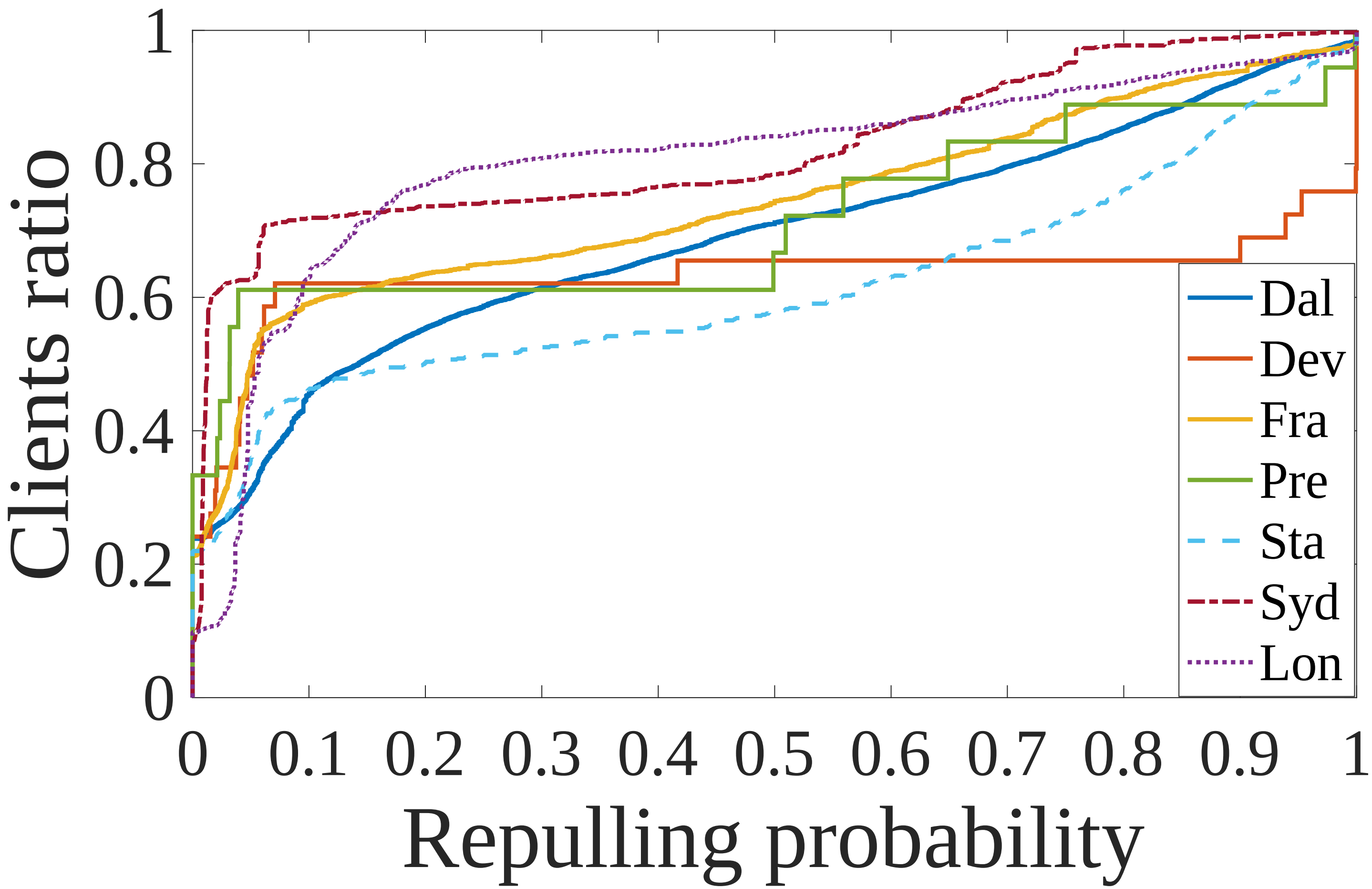
Key observation II: Predictable user access pattern

- We observe a consistent user pulling pattern: Pull manifest first, then layers, but not all of the layers will be pulled.
- We performed a quantitive study using a 75-day IBM Cloud Registry workload with 7 availability zones.

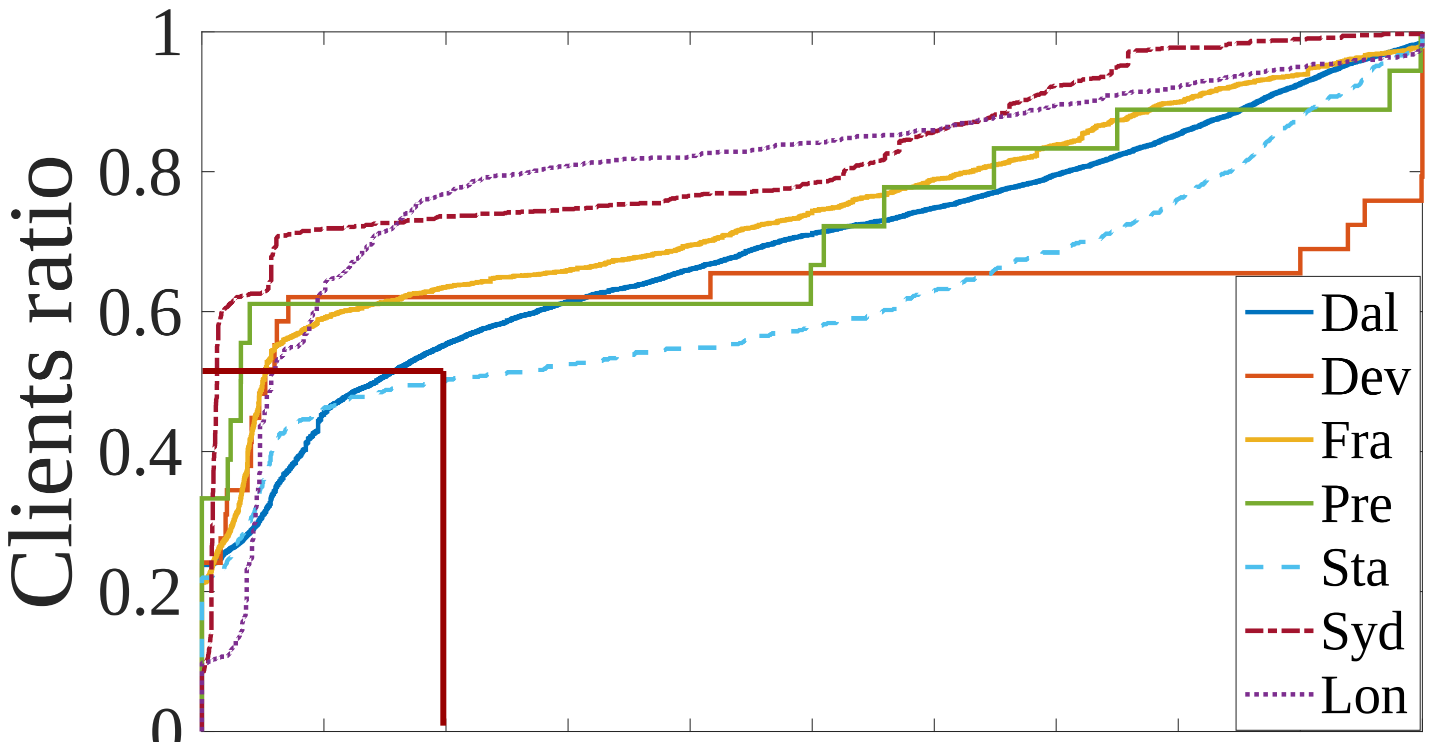


Majority of layers are only fetched once by the same client.

Key observation II-b: User repulling pattern can also be predicted

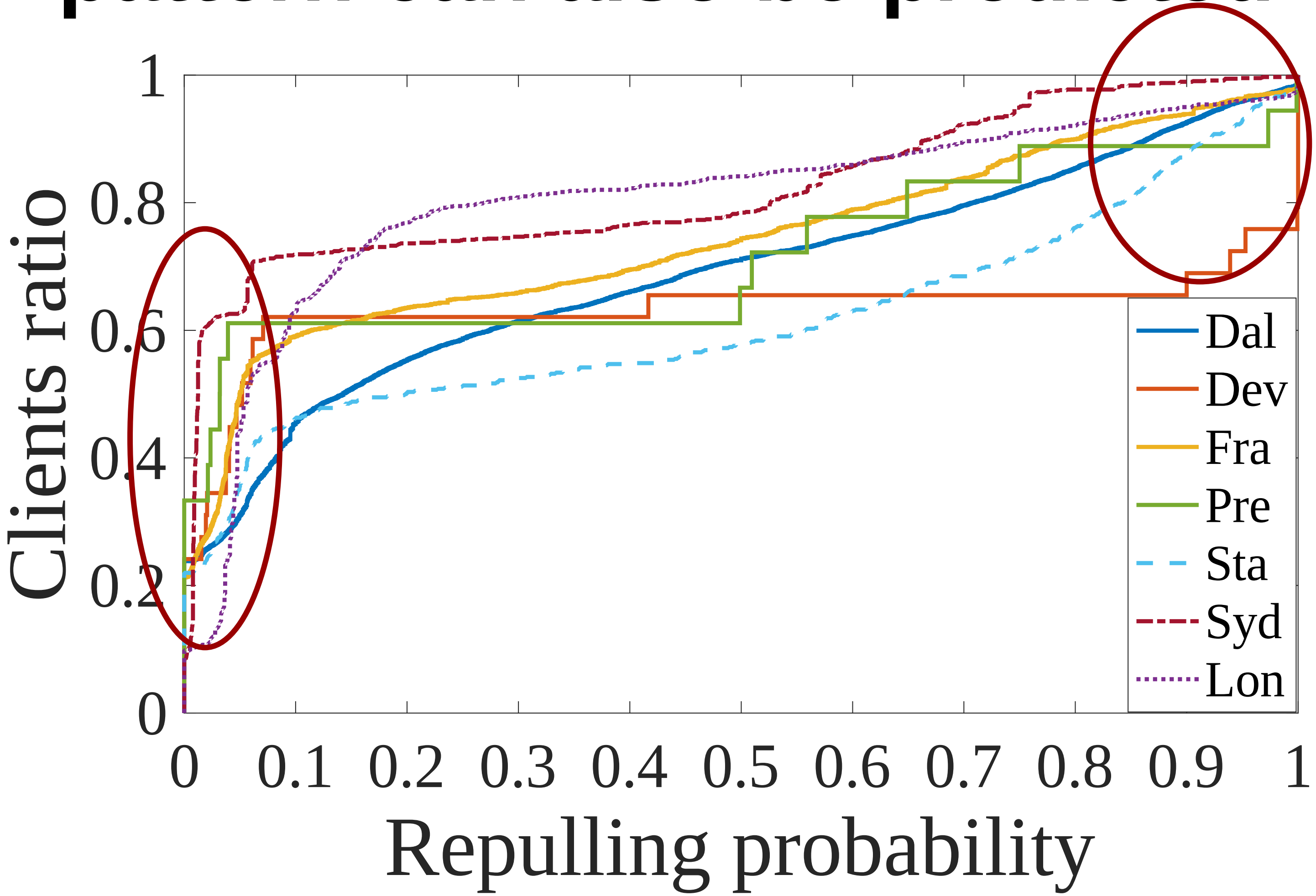


Key observation II-b: User repulling pattern can also be predicted

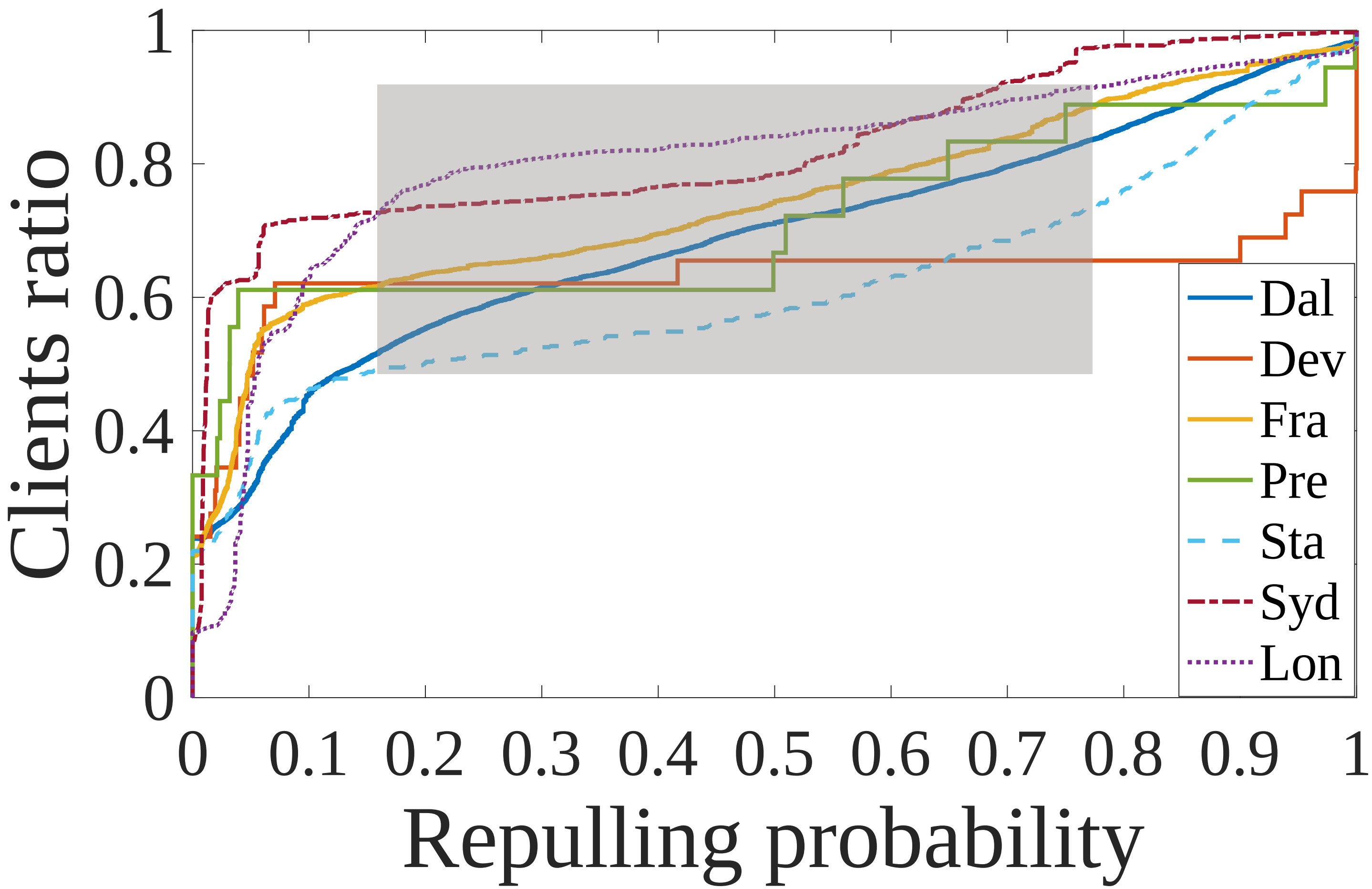


Half of the clients have a repull probability less than 0.2 → many clients pull a layer only once.

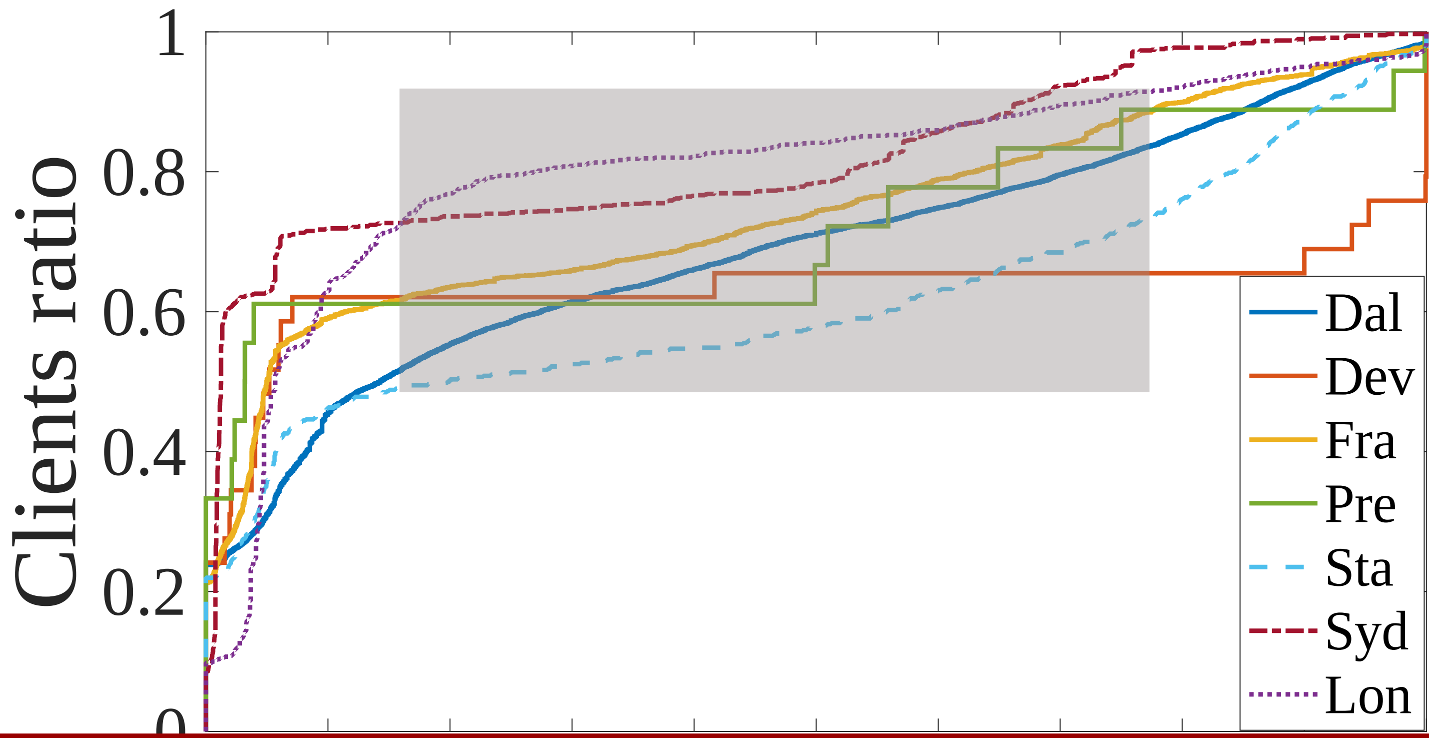
Key observation II-b: User repulling pattern can also be predicted



Key observation II-b: User repulling pattern can also be predicted

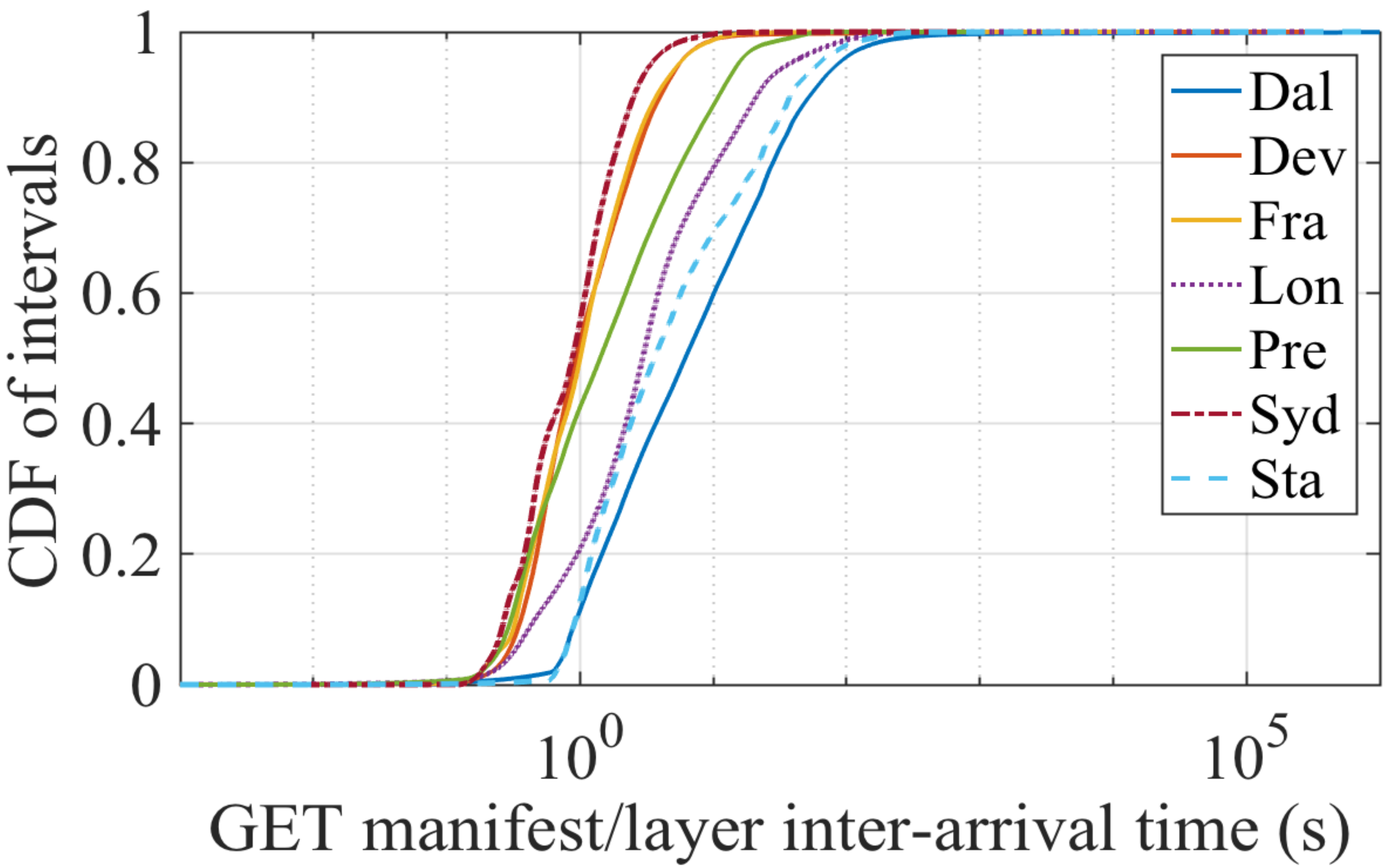


Key observation II-b: User repulling pattern can also be predicted

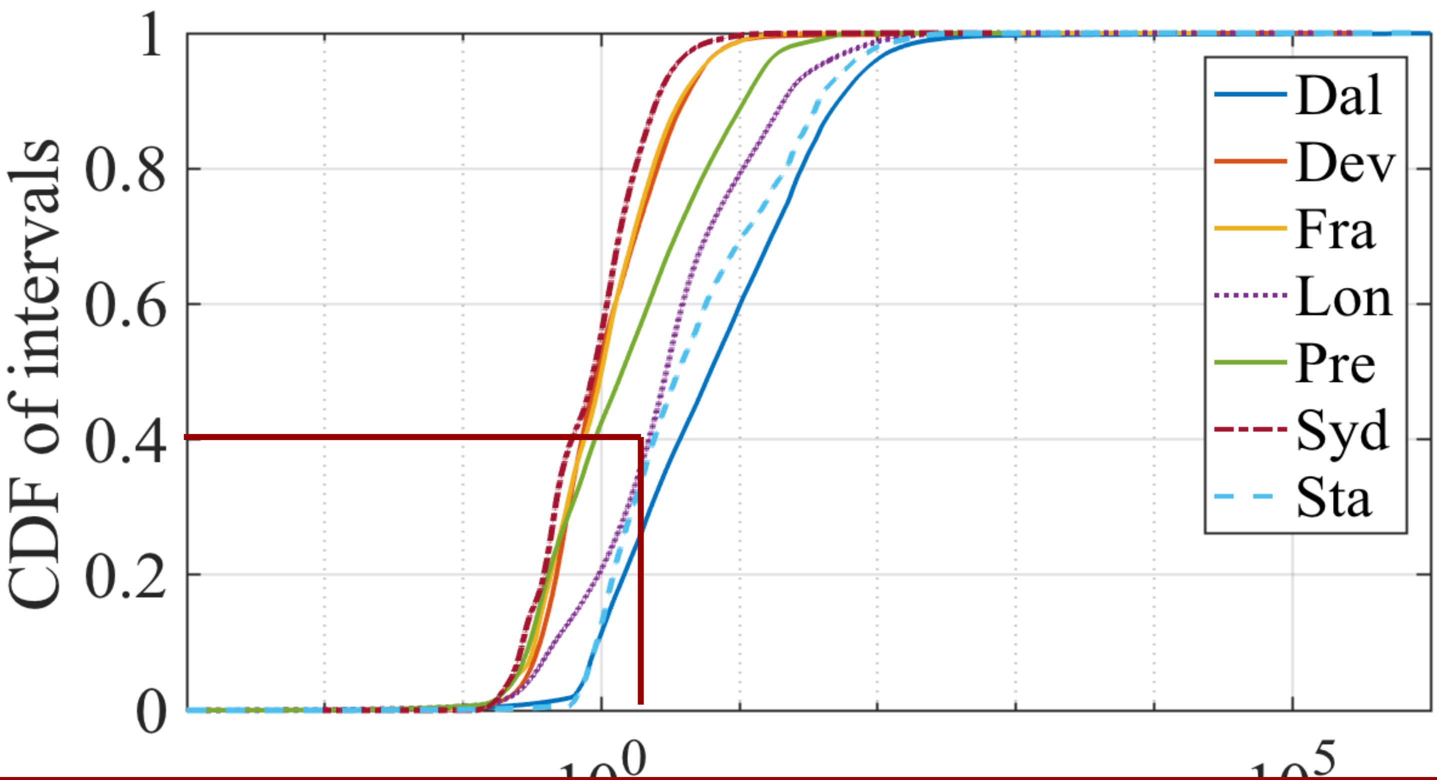


User repulling pattern is either pull-once or always-pull → we can predict which layers to pull.

Key observation II-c: Layer preconstruction is possible

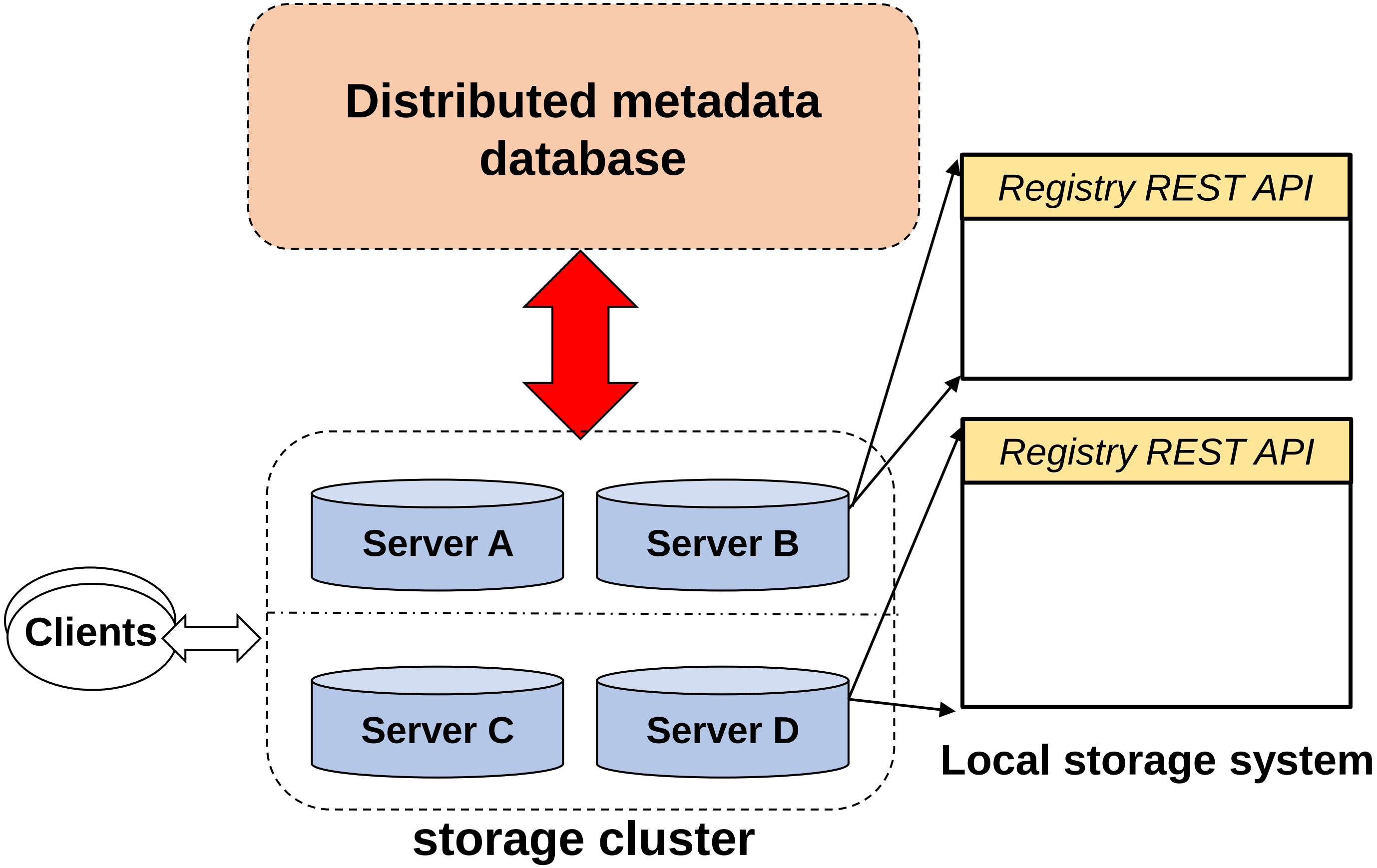


Key observation II-c: Layer preconstruction is possible



Layer preconstruction can significantly reduce layer restore overhead.

DupHunter architecture



Reducing overhead in DupHunter

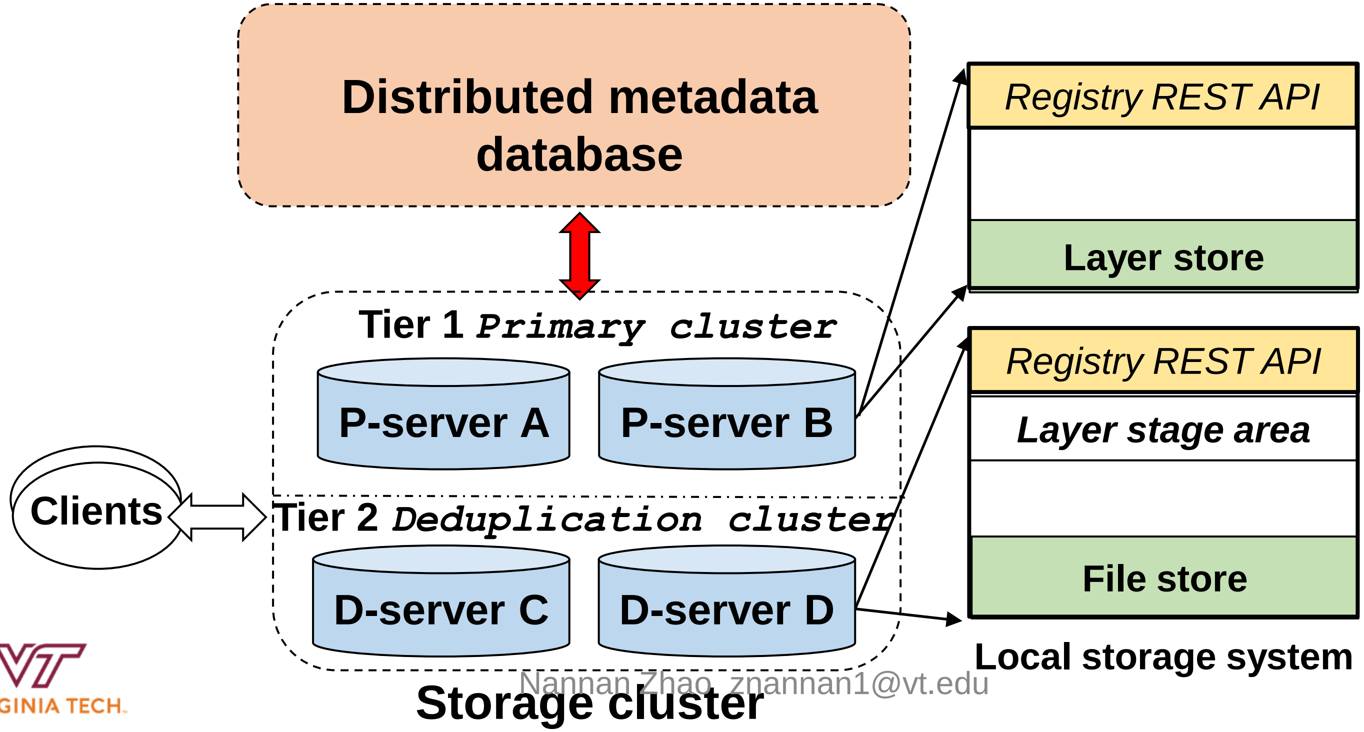
1. Support multiple replica deduplication modes.
2. Facilitate parallel layer reconstruction.
3. Enable proactive layer prefetching/preconstruction.

DupHunter supports multiple replica deduplication modes

- **B-mode *n***: Basic deduplication mode *n*
  - Keep *n* layer replicas intact.
  - Deduplicate the remaining *R-n* layer replicas (*R* = layer replication level).
- **S-mode**: Selective deduplication mode
  - The number of intact layer replicas proportional to the layer's popularity.
  - Hot layers have more intact replicas.

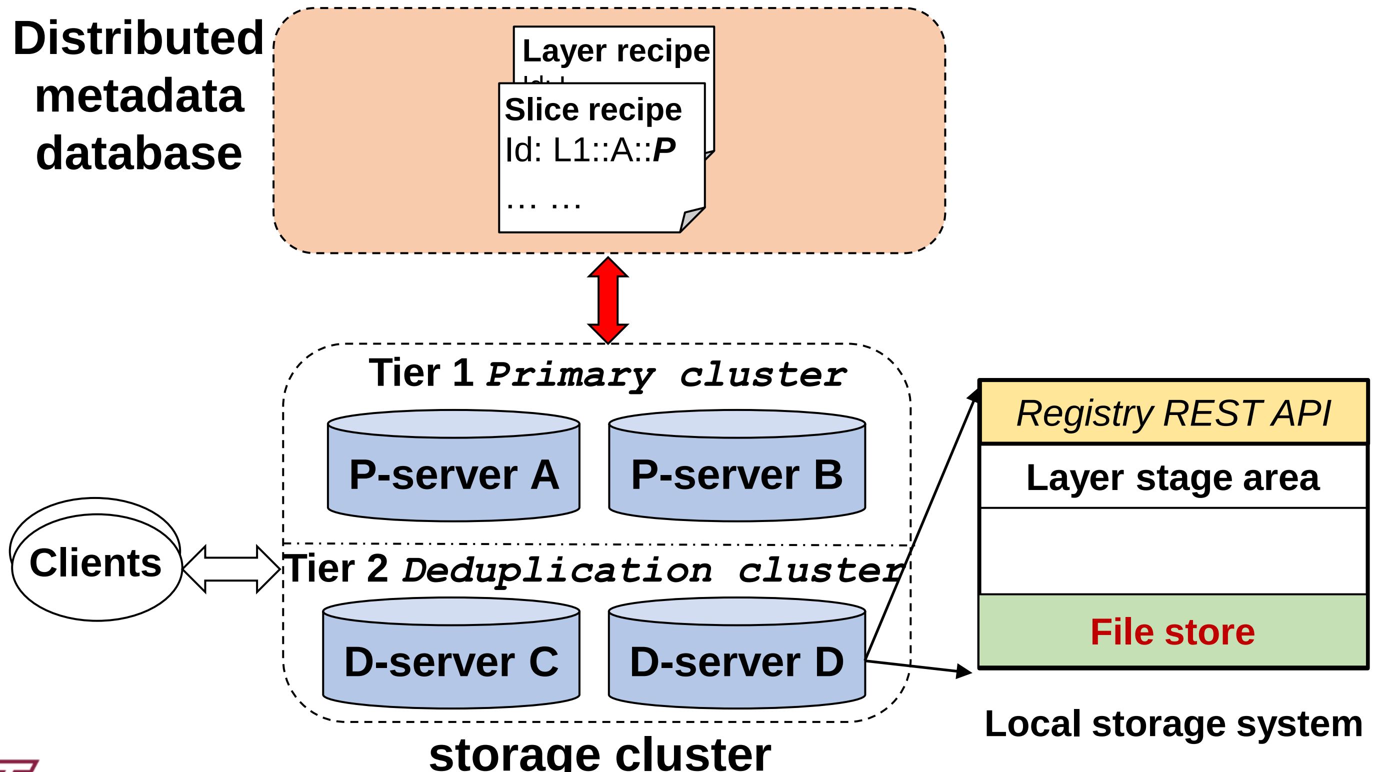
DupHunter supports multiple replica deduplication modes

- **B-mode *n***: Basic deduplication mode *n*
  - Keep *n* layer replicas intact.
  - Deduplicate the remaining *R-n* layer replicas (*R* = layer replication level).
- **S-mode**: Selective deduplication mode
  - The number of intact layer replicas proportional to the layer's popularity.
  - Hot layers have more intact replicas.



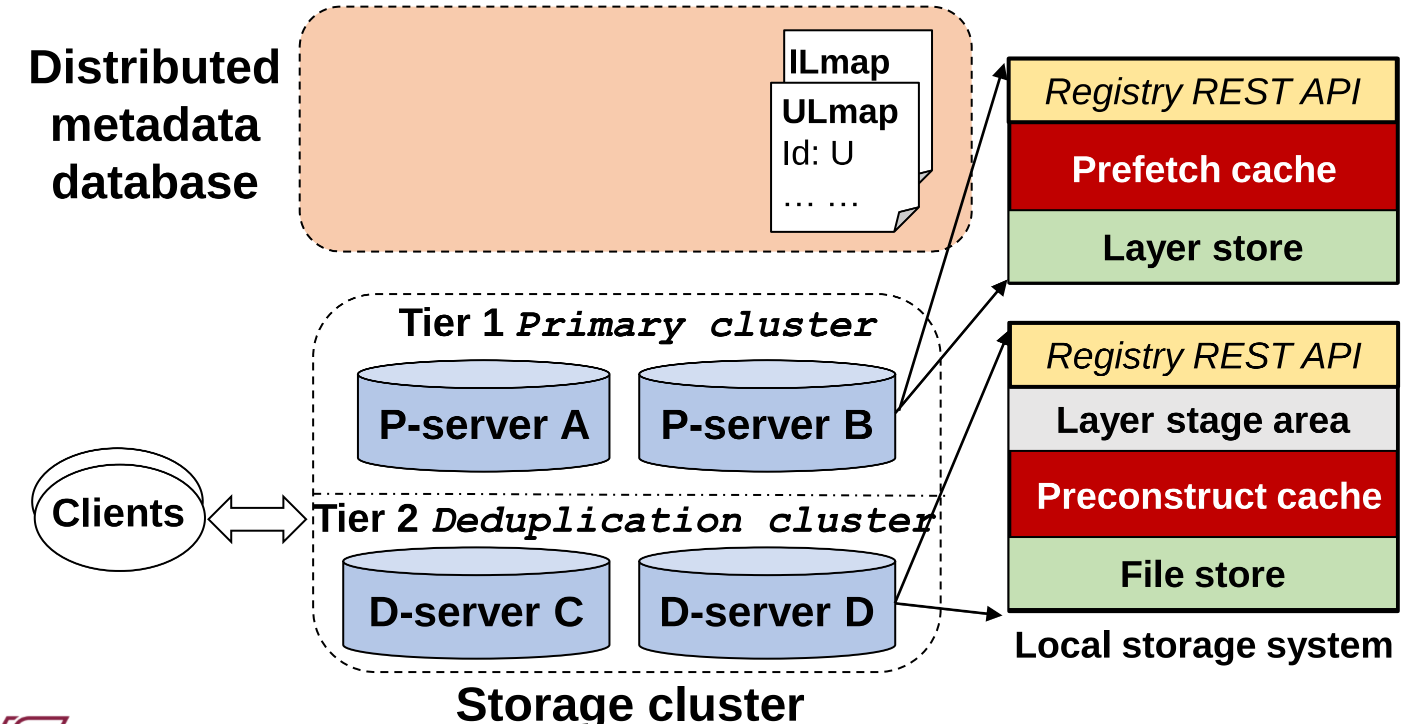
DupHunter facilitates parallel layer reconstruction

- **Slice**: Set of all the files on a server belonging to a layer.
  - Distributed evenly across the cluster.
  - Speed up layer reconstruction via parallel processing of slices.

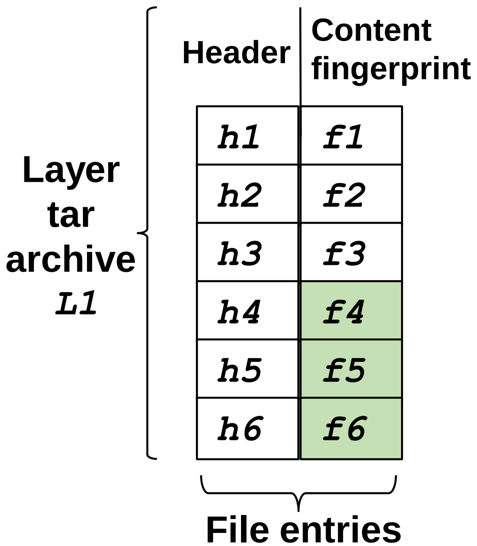


DupHunter enables prefetching/preconstruction of layers

- **Prefetch cache** to prefetch layers and hide disk I/Os.
- **Preconstruct cache** to store preconstruct layers and hide layer restore overhead.



Deduplicating layers



Deduplicating layers

Layer tar archive *L1*

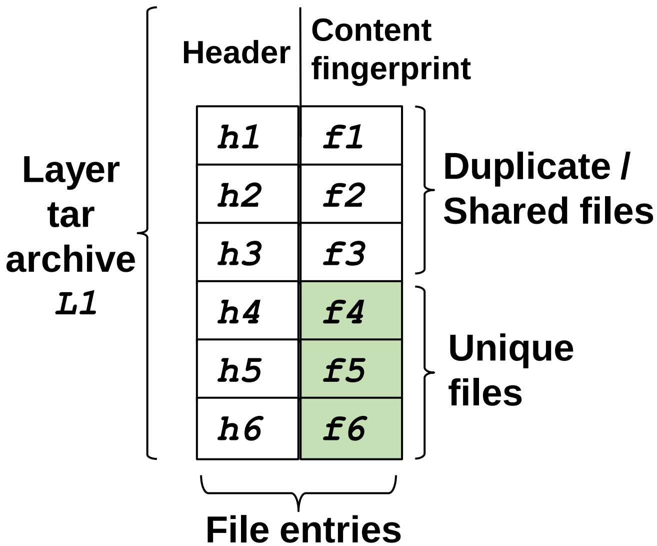
Header	Content fingerprint
h1	*f1*
h2	*f2*
h3	*f3*
h4	*f4*
h5	*f5*
h6	*f6*

File entries

File index

Id	*r1*	*r2*
f1	A:/../..	B:/../..
f2	B:/../..	C:/../..

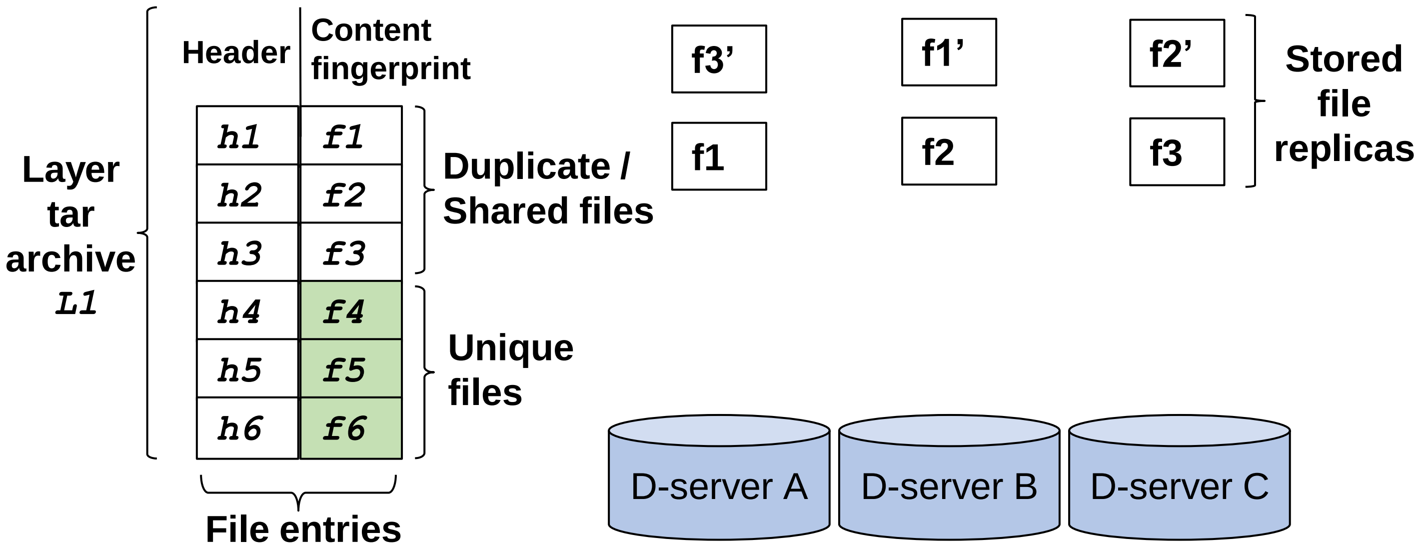
Deduplicating layers



File index

Id	*r1*	*r2*
f1	A:/../..	B:/../..
f2	B:/../..	C:/../..

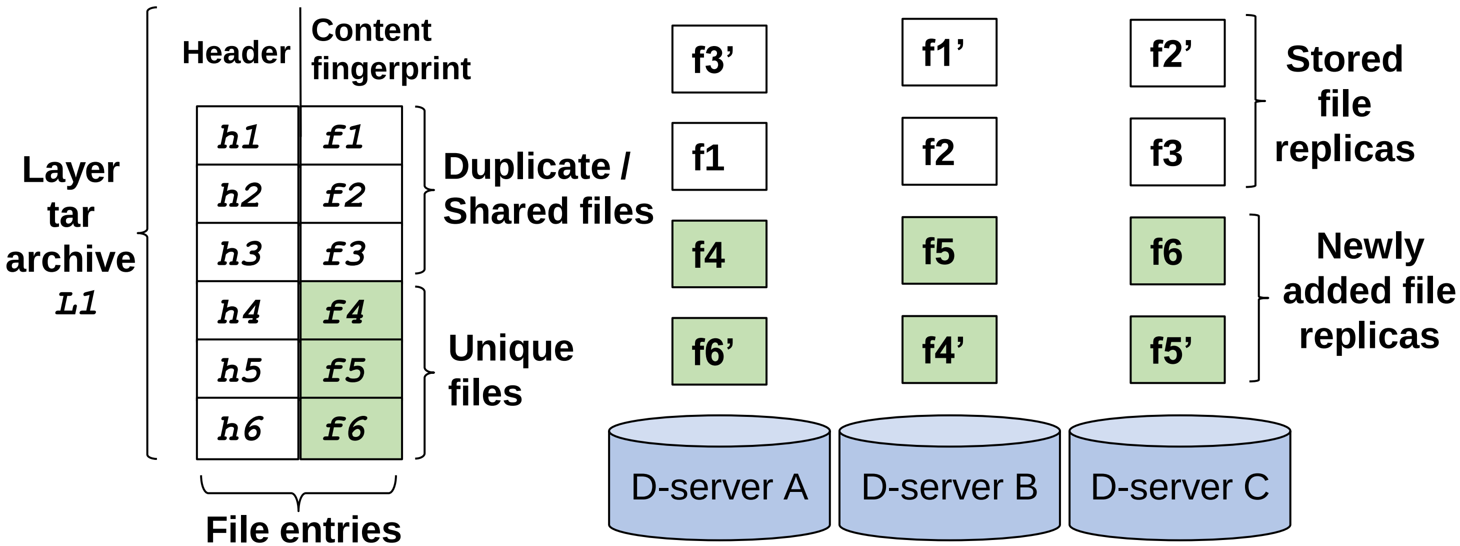
Deduplicating layers



File index

Id	*r1*	*r2*
f1	A:/../..	B:/../..
f2	B:/../..	C:/../..

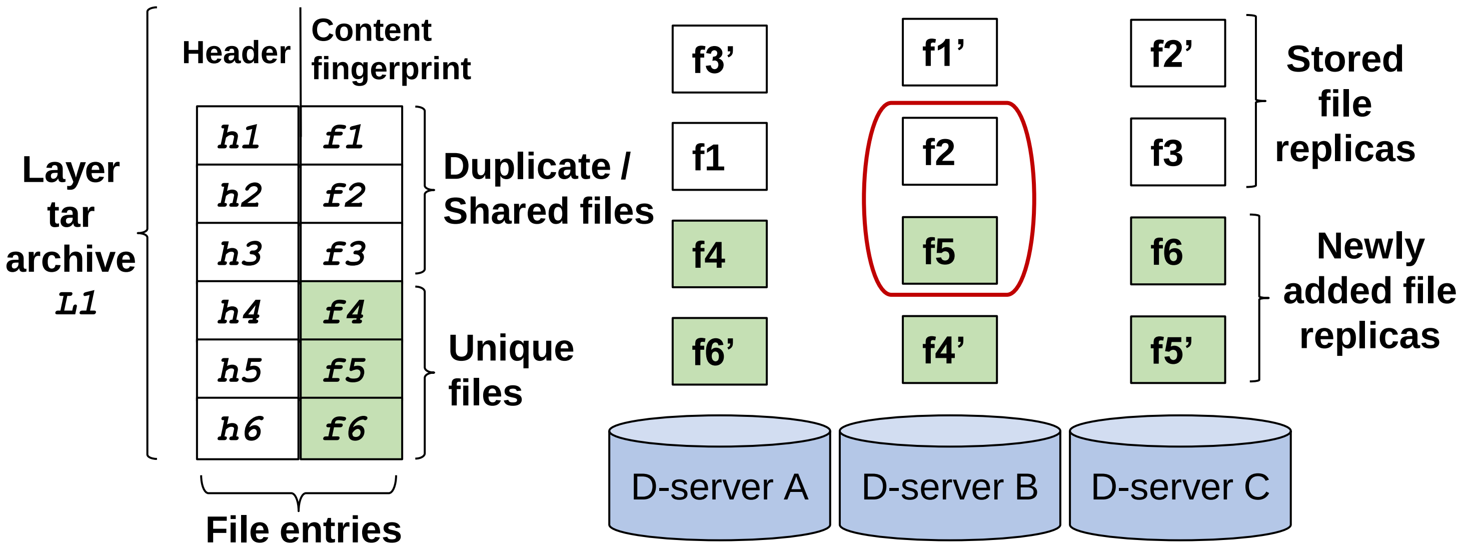
Deduplicating layers



File index

Id	r1	r2
f1	A:/../..	B:/../..
f2	B:/../..	C:/../..

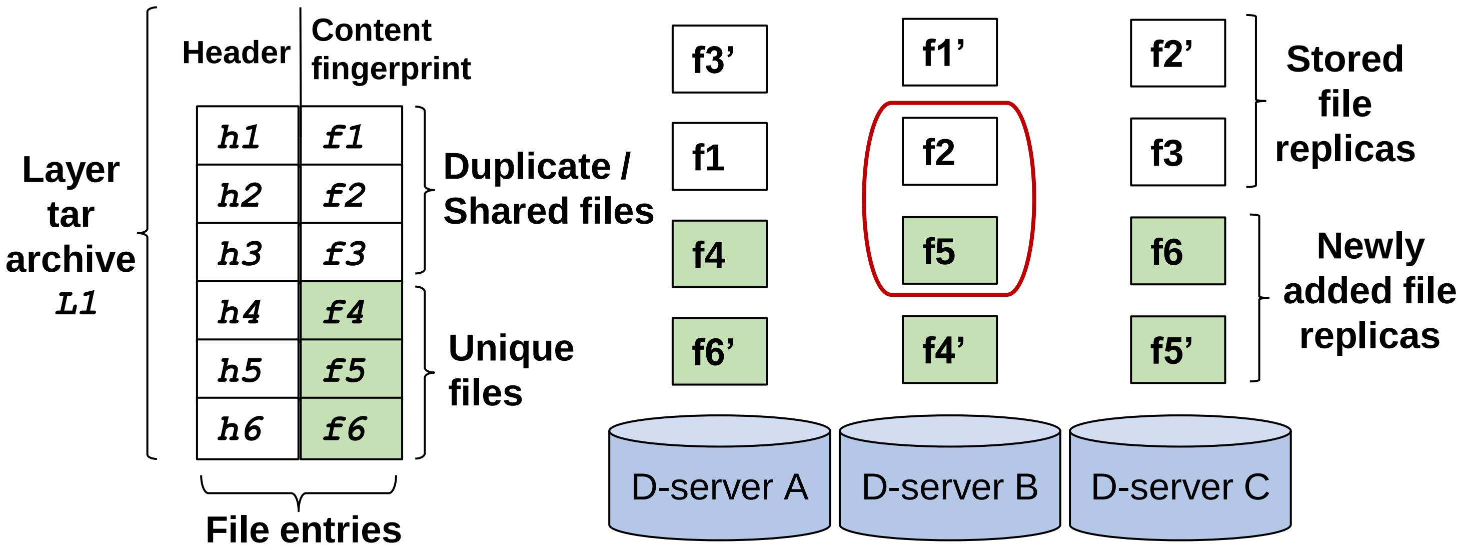
Deduplicating layers



File index

Id	*r1*	*r2*
f1	A:/../..	B:/../..
f2	B:/../..	C:/../..

Deduplicating layers



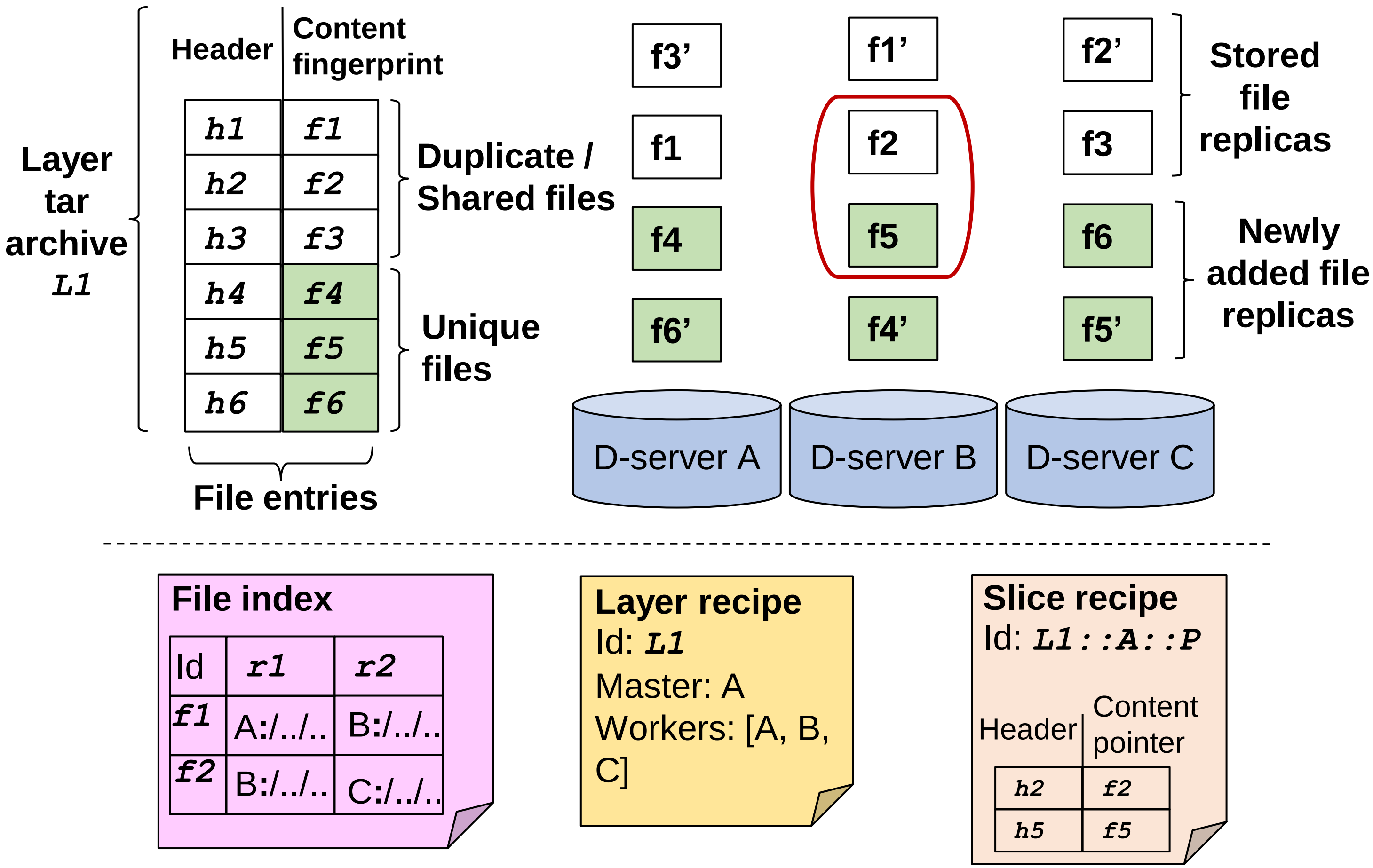
File index

Id	*r1*	*r2*
f1	A:/../..	B:/../..
f2	B:/../..	C:/../..

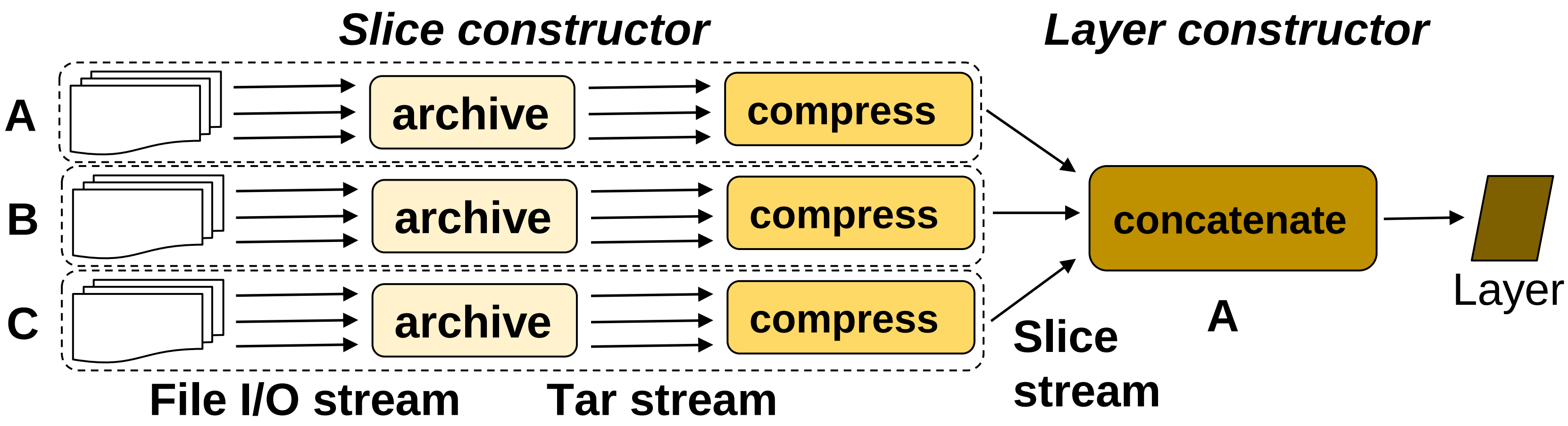
Slice recipe
Id: ***L1::A::P***

Header	Content pointer
h2	*f2*
h5	*f5*

Deduplicating layers



Restoring layers



Caching and preconstructing layers

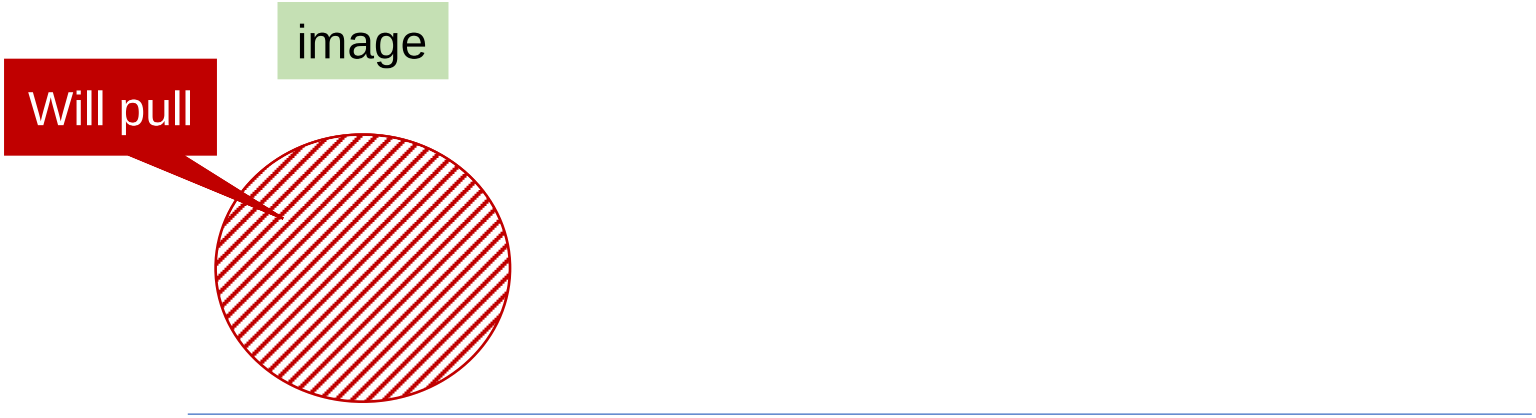
- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.

Caching and preconstructing layers

- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.

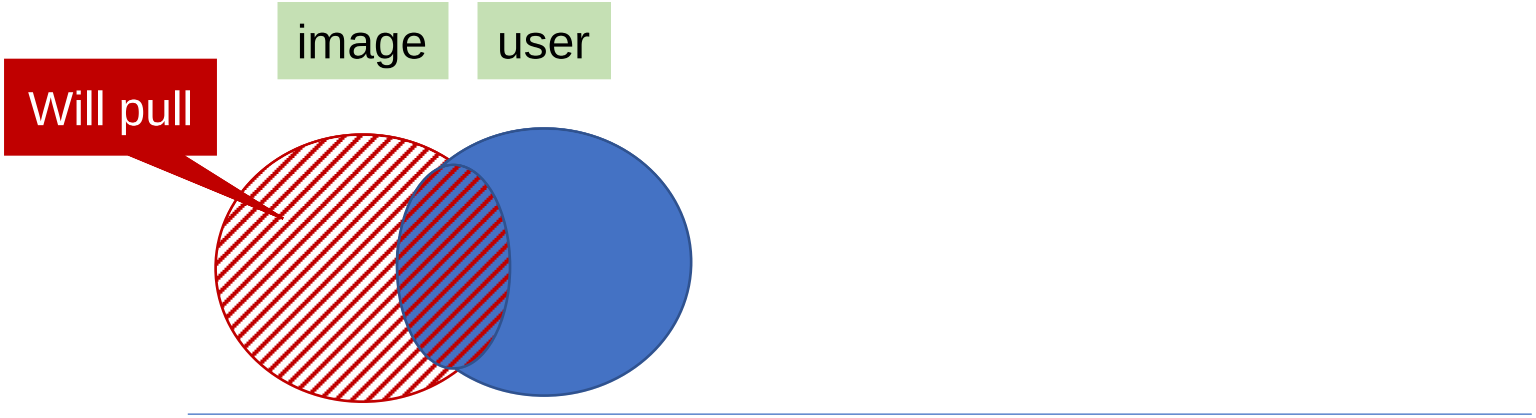
Caching and preconstructing layers

- ❑ ILmap: Maps image to its containing layer set.
- ❑ ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.



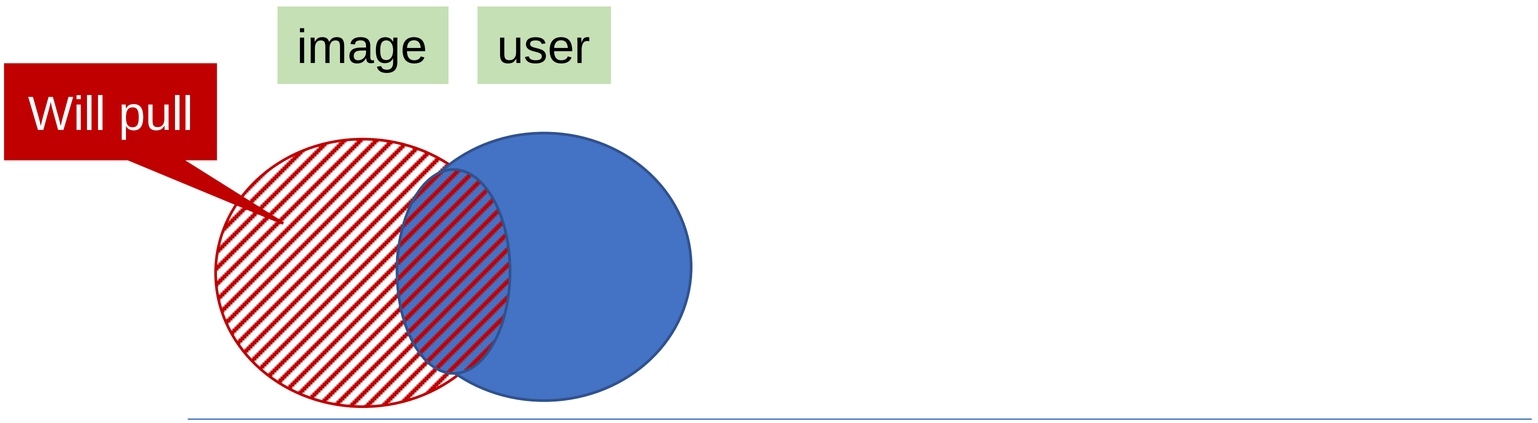
Caching and preconstructing layers

- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.



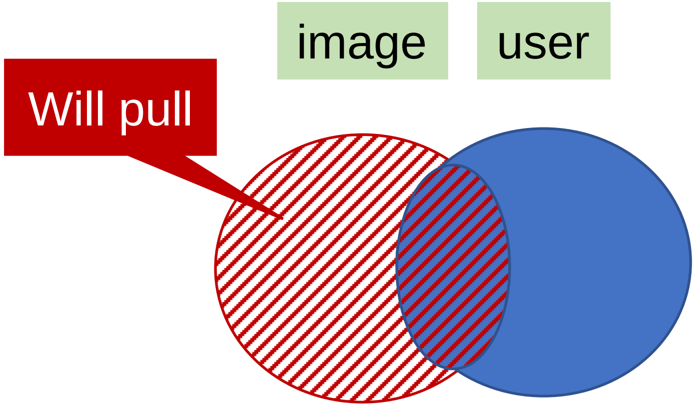
Caching and preconstructing layers

- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.



Caching and preconstructing layers

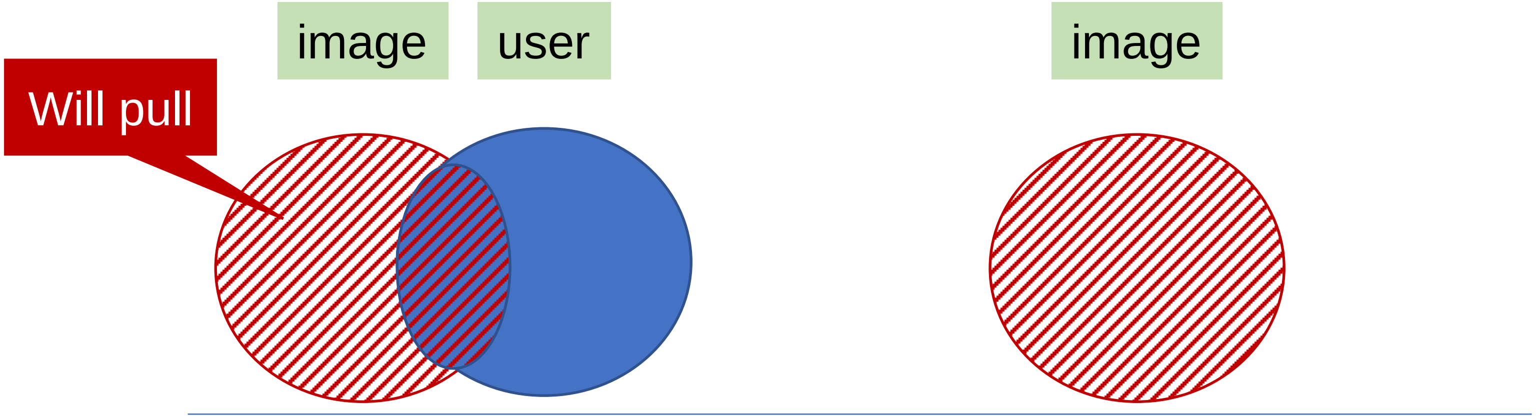
- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.



$$S_\Delta = ILmap[r.img] - ULmap[r.addr]$$

Caching and preconstructing layers

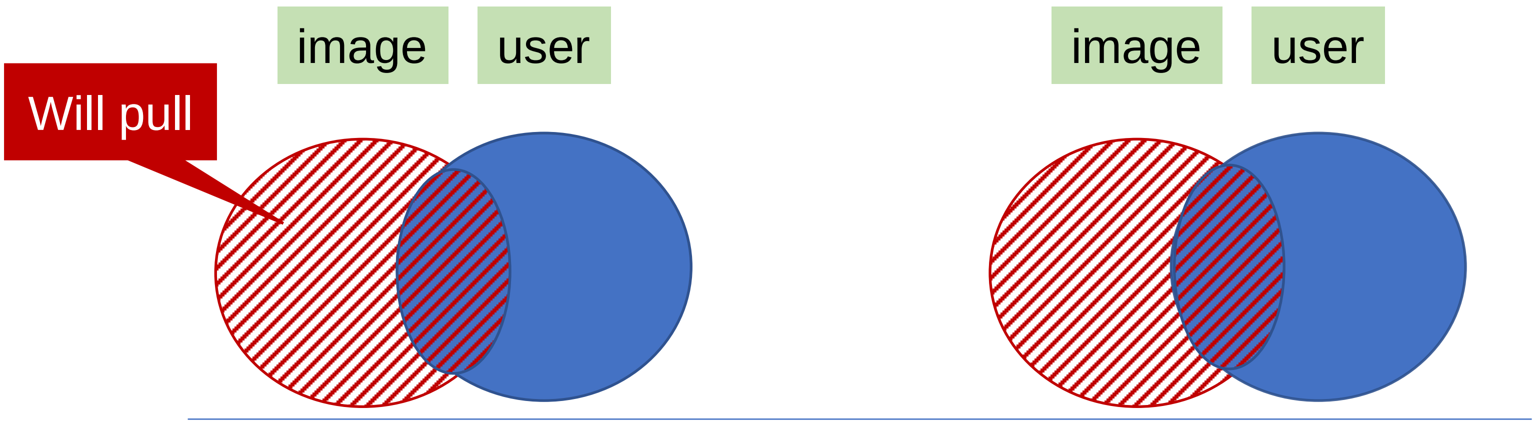
- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.



$$S_{\Delta} = ILmap[r.img] - ULmap[r.addr]$$

Caching and preconstructing layers

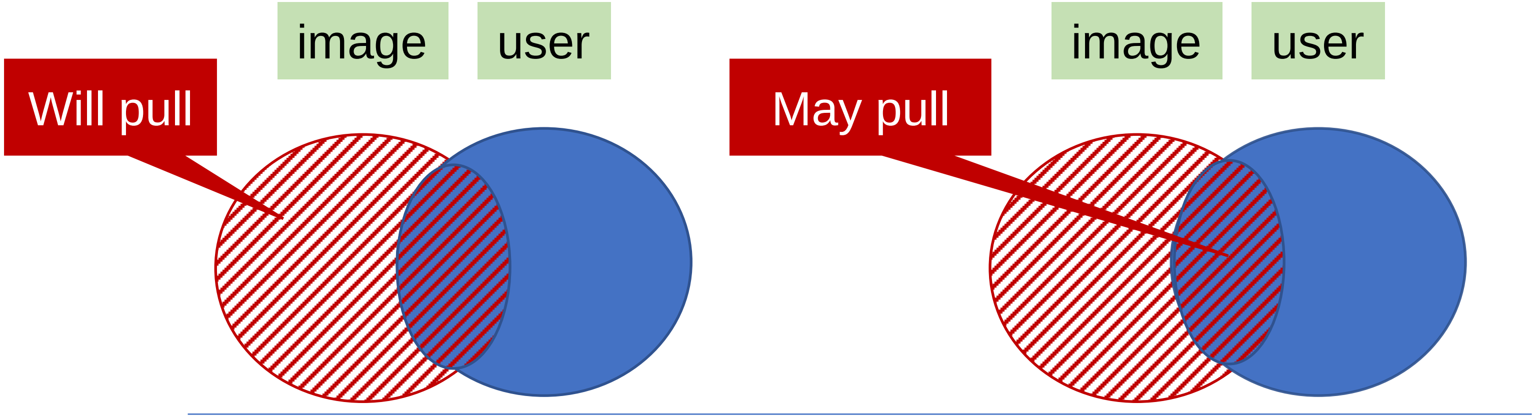
- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.



$$S_\Delta = ILmap[r.img] - ULmap[r.addr]$$

Caching and preconstructing layers

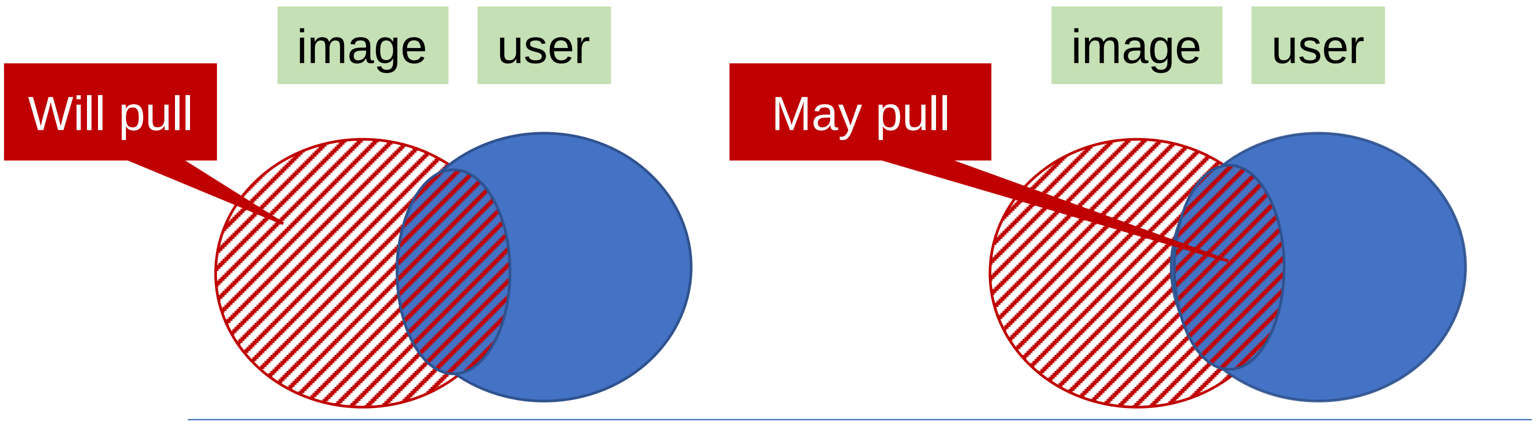
- ❑ ILmap: Maps image to its containing layer set.
- ❑ ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.



$$S_{\Delta} = ILmap[r.img] - ULmap[r.addr]$$

Caching and preconstructing layers

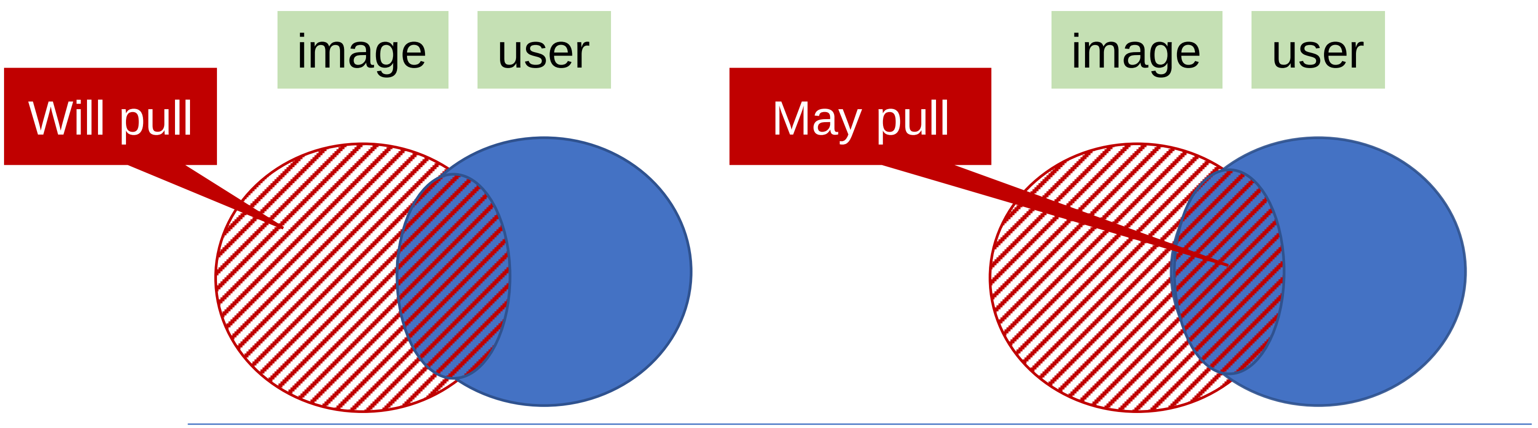
- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.



$$S_{\Delta} = ILmap[r.img] - ULmap[r.addr]$$

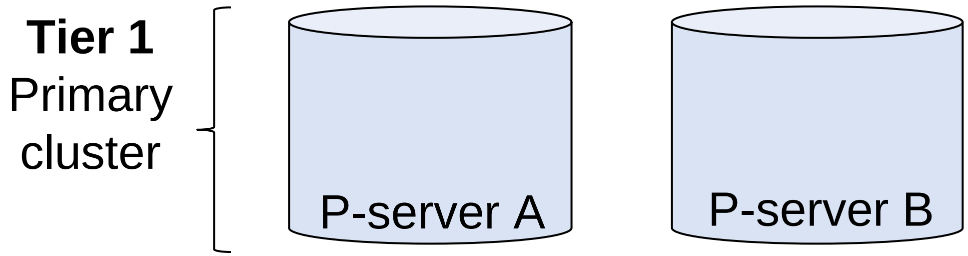
Caching and preconstructing layers

- ILmap: Maps image to its containing layer set.
- ULmap: Maps user to the layers that the user has accessed and the corresponding pull count.

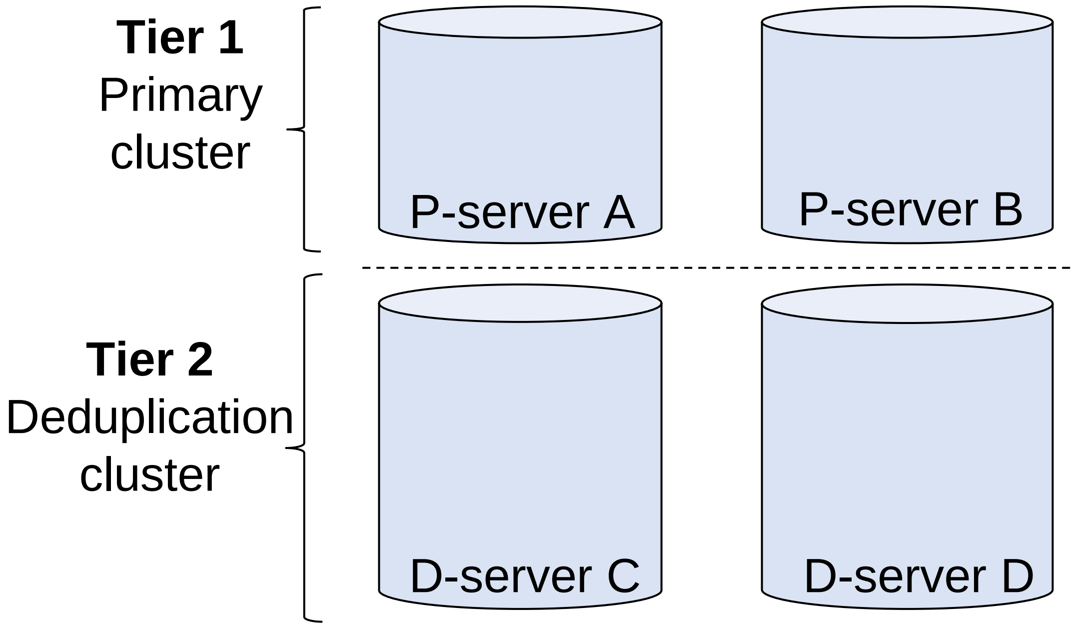


$$S_\Delta = ILmap[r.img] - ULmap[r.addr] \qquad S_\cap = ILmap[r.img] \cap ULmap[r.addr]$$

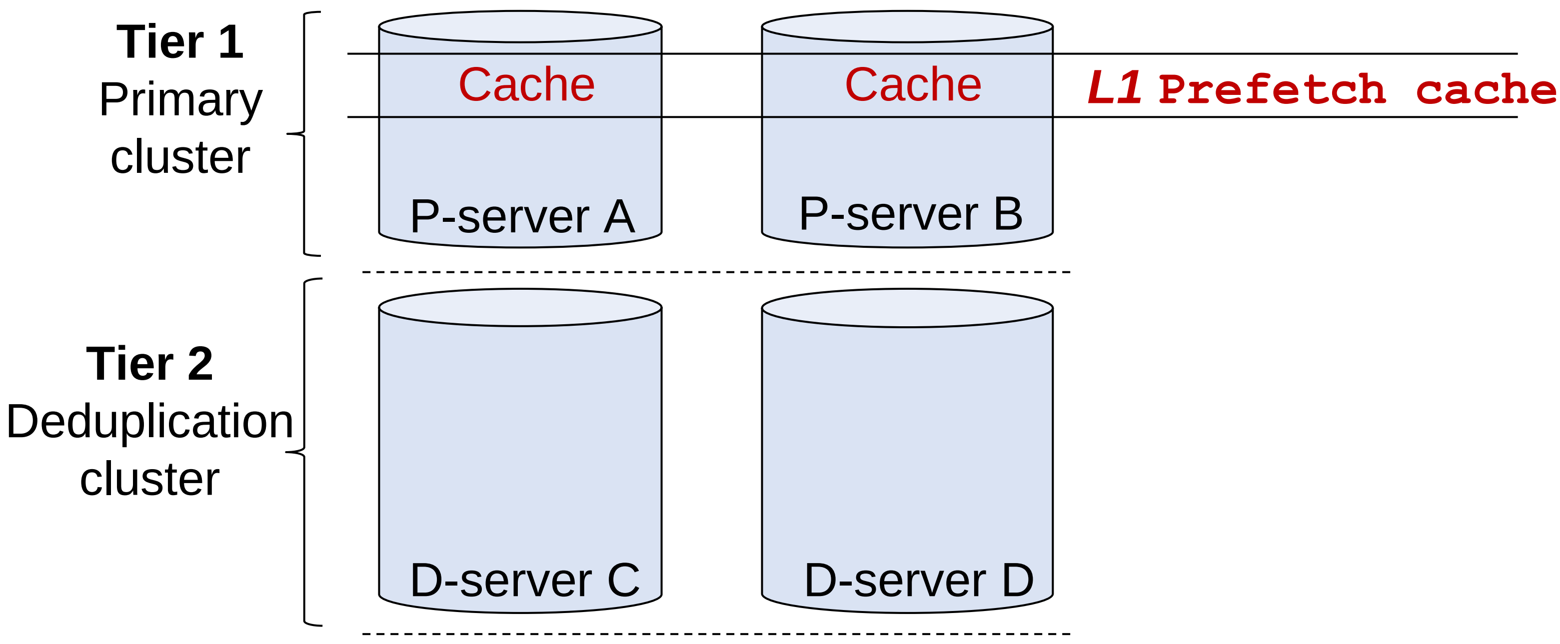
Cache handling in tiered storage



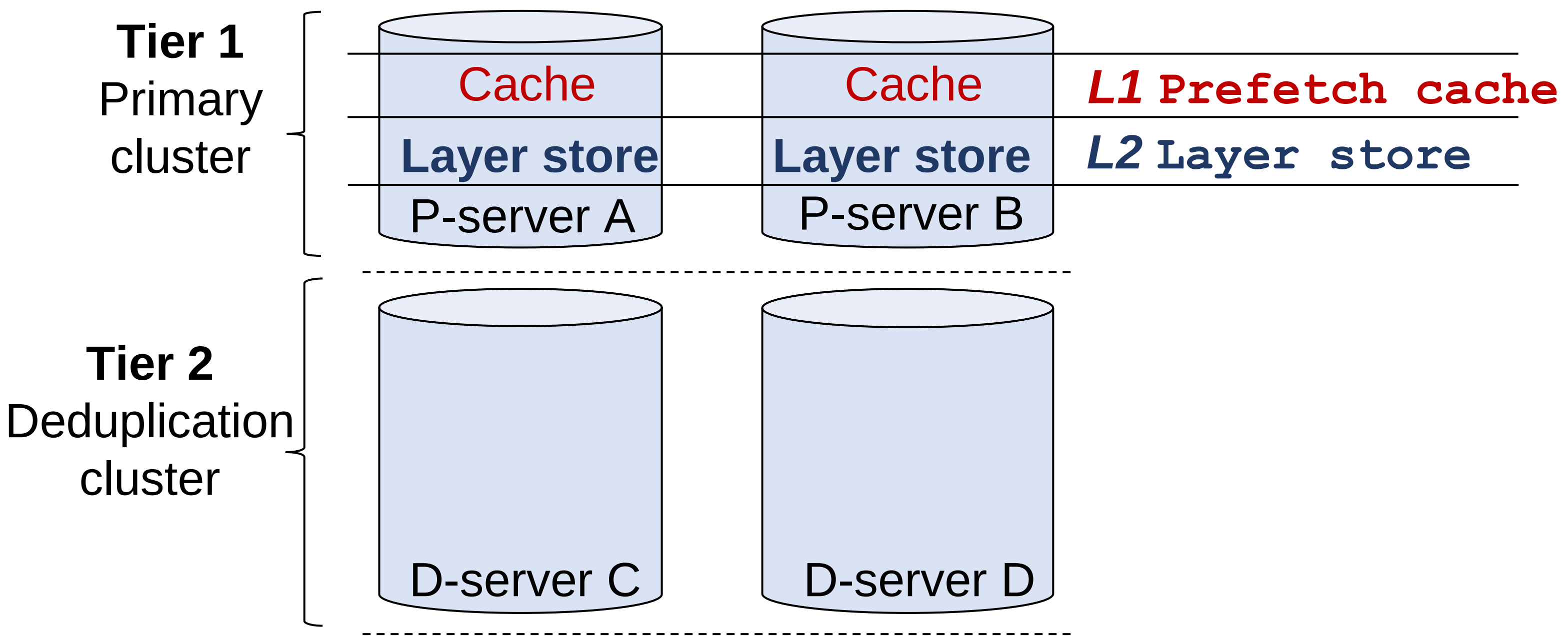
Cache handling in tiered storage



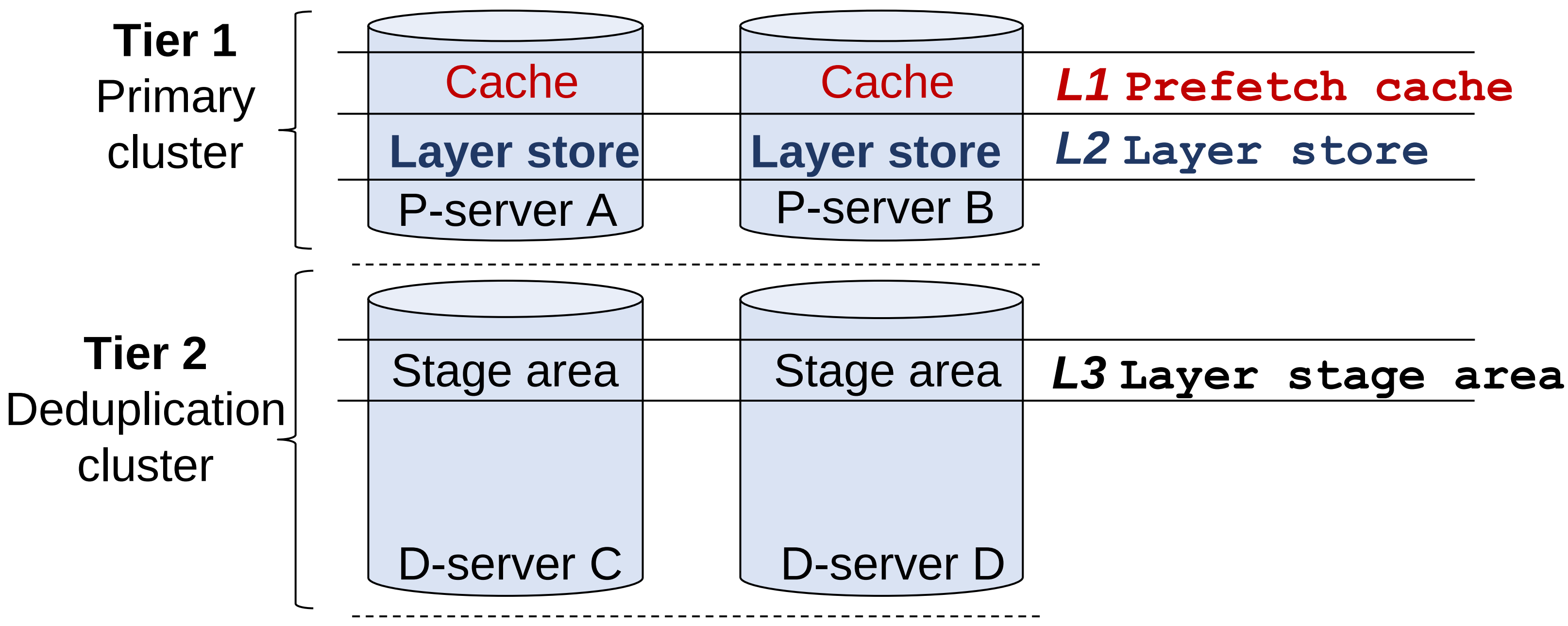
Cache handling in tiered storage



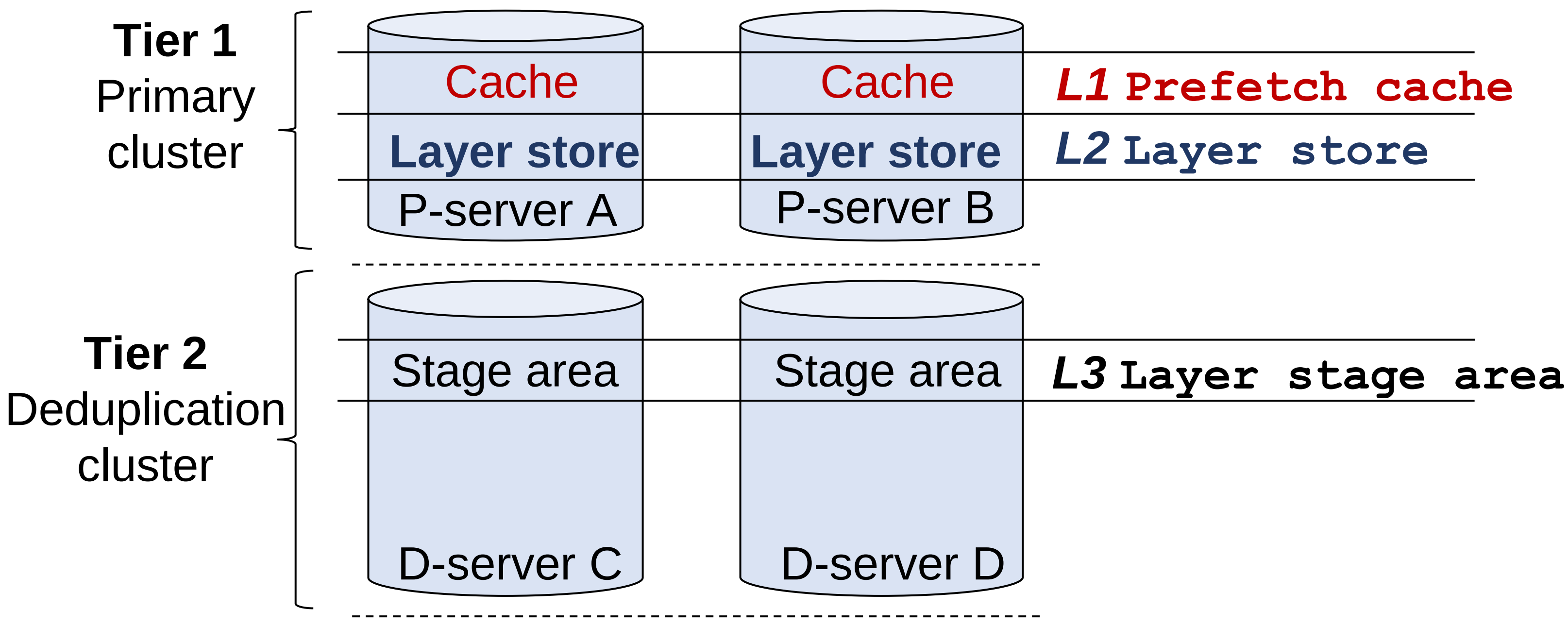
Cache handling in tiered storage



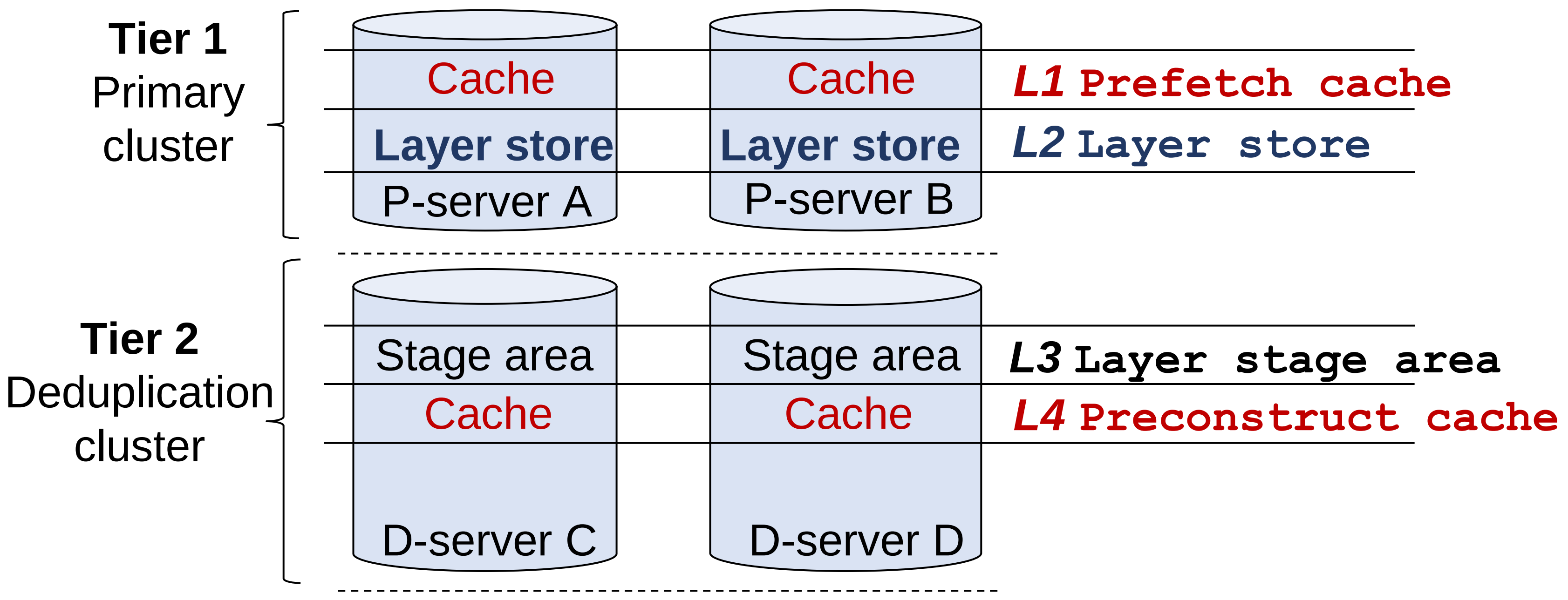
Cache handling in tiered storage



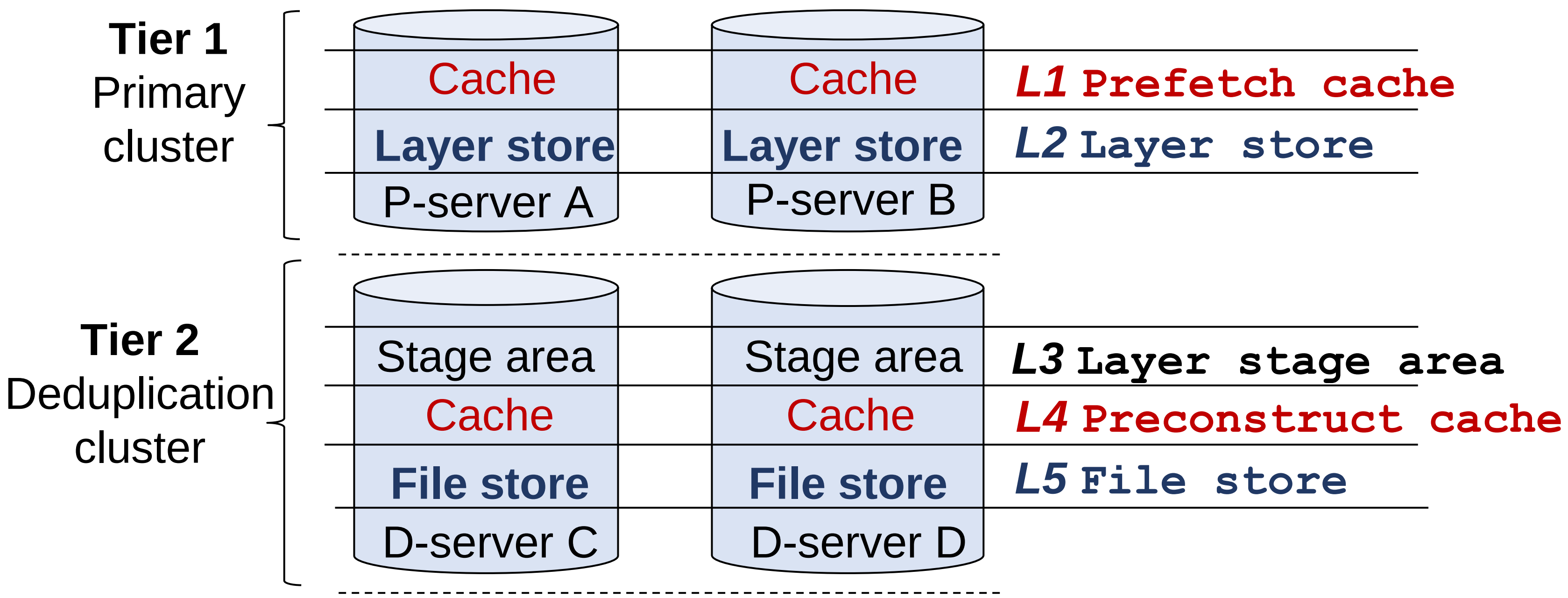
Cache handling in tiered storage



Cache handling in tiered storage



Cache handling in tiered storage



Evaluation

- Workloads used:
  - Traces from IBM registries: Dal, Fra, Lon, and Syd availability zones
  - Dataset from Docker Hub

- Schemes studied:
  - **Baseline**: No deduplication
  - **B-mode *n*:** *n (1-3)* replicas are preserved; *3 – n* deduplicated
  - **S-mode**: intact layer replicas proportional to the layer's popularity
  - **B-mode 0**: deduplicate all layer replicas, under a given replication policy
    - **GF-R**: global file-level deduplication
    - **GF+LB-R**: global file-level deduplication and local block-level deduplication
    - **GB-EC**: global block-level deduplication under erasure coding

Deduplication ratio vs. performance

Mode	Dedup. ratio	Performance improvement (P-servers)
B-mode 1	1.5	1.6×
S-mode	1.3	2×
B-mode 2	1.2	2.6×
B-mode 3	1	2.8×

	Dedup ratio	Performance degradation (D-servers)
B-mode 0	**GF-R** (Global file-level [3 replicas])	
	2.1	-1.03 ×
	GF+LB-R (Global file- and local block-level [3 replicas])	
	3.0	-2.87×
	GB-EC (Global block-level [Erasure coding])	
	6.9	-6.37×

VIRGINIA TECH.

Deduplication ratio vs. performance

Mode	Dedup. ratio	Performance improvement (P-servers)
B-mode 1	1.5	1.6×
S-mode	1.3	2×
B-mode 2	1.2	2.6×
B-mode 3	1	2.8×

	Dedup ratio	Performance degradation (D-servers)
B-mode 0	**GF-R** (Global file-level [3 replicas])	
	2.1	-1.03 ×
	GF+LB-R (Global file- and local block-level [3 replicas])	
	3.0	-2.87×
	GB-EC (Global block-level [Erasure coding])	
	6.9	-6.37×

Deduplication ratio vs. performance

Mode	Dedup. ratio	Performance improvement (P-servers)
B-mode 1	1.5	1.6×
S-mode	1.3	2×
B-mode 2	1.2	2.6×
B-mode 3	1	2.8×

	Dedup ratio	Performance degradation (D-servers)
B-mode 0	**GF-R** (Global file-level [3 replicas])	
	2.1	-1.03 ×
	GF+LB-R (Global file- and local block-level [3 replicas])	
	3.0	-2.87×
	GB-EC (Global block-level [Erasure coding])	
	6.9	-6.37×

Deduplication ratio vs. performance

Mode	Dedup. ratio	Performance improvement (P-servers)
B-mode 1	1.5	1.6×
S-mode	1.3	2×
B-mode 2	1.2	2.6×
B-mode 3	1	2.8×

	Dedup ratio	Performance degradation (D-servers)
B-mode 0	**GF-R** (Global file-level [3 replicas])	
	2.1	-1.03 ×
	GF+LB-R (Global file- and local block-level [3 replicas])	
	3.0	-2.87×
	GB-EC (Global block-level [Erasure coding])	
	6.9	-6.37×

Deduplication ratio vs. performance

Mode	Dedup. ratio	Performance improvement (P-servers)
B-mode 1	1.5	1.6×
S-mode	1.3	2×
B-mode 2	1.2	2.6×
B-mode 3	1	2.8×

	Dedup ratio	Performance degradation (D-servers)
B-mode 0	**GF-R** (Global file-level [3 replicas])	
	2.1	-1.03 ×
	GF+LB-R (Global file- and local block-level [3 replicas])	
	3.0	-2.87×
	GB-EC (Global block-level [Erasure coding])	
	6.9	-6.37×

Deduplication ratio vs. performance

Mode	Dedup. ratio	Performance improvement (P-servers)
B-mode 1	1.5	1.6×
S-mode	1.3	2×
B-mode 2	1.2	2.6×
B-mode 3	1	2.8×

	Dedup ratio	Performance degradation (D-servers)
B-mode 0	**GF-R** (Global file-level [3 replicas])	
	2.1	-1.03 ×
	GF+LB-R (Global file- and local block-level [3 replicas])	
	3.0	-2.87×
	GB-EC (Global block-level [Erasure coding])	
	6.9	-6.37×

Deduplication ratio vs. performance

Mode	Dedup. ratio	Performance improvement (P-servers)
B-mode 1	1.5	1.6×
S-mode	1.3	2×
B-mode 2	1.2	2.6×
B-mode 3	1	2.8×

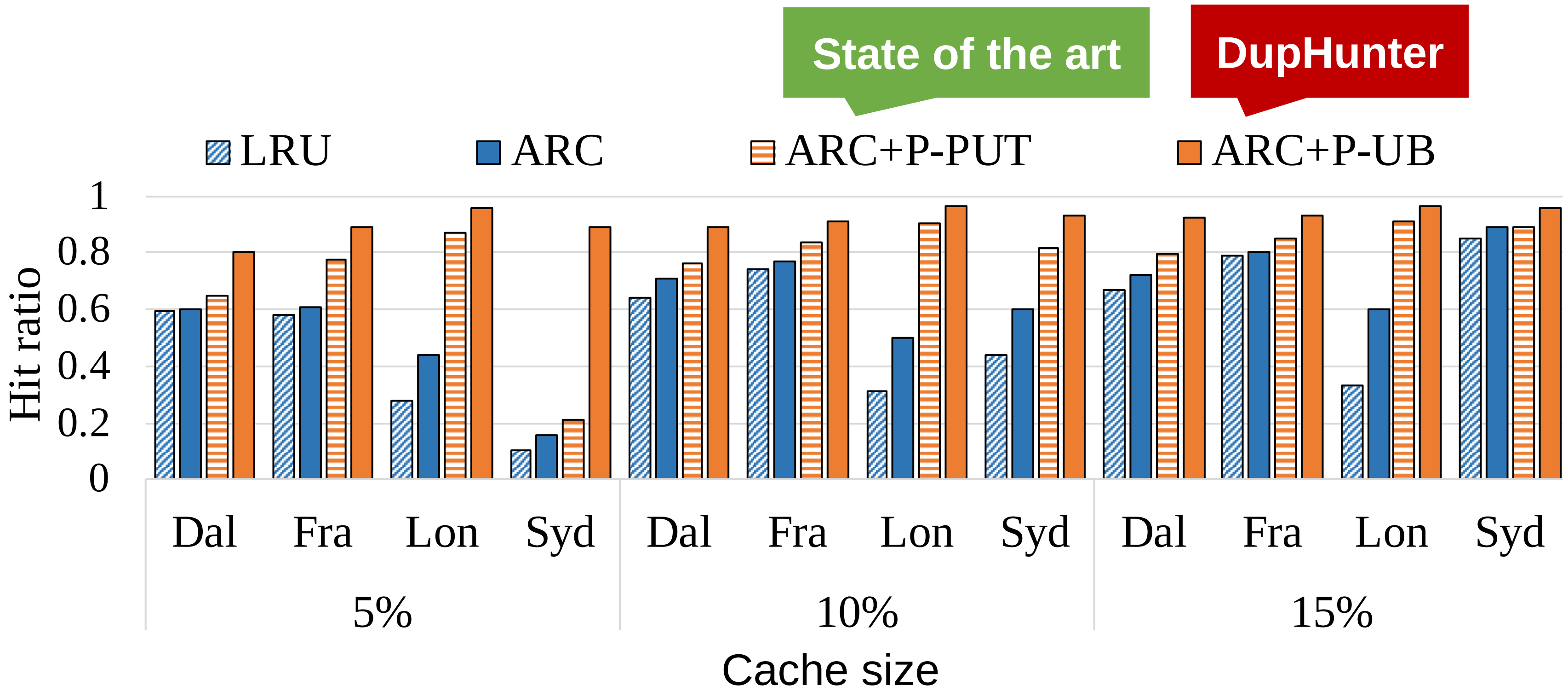
	Dedup ratio	Performance degradation (D-servers)
B-mode 0	**GF-R** (Global file-level [3 replicas])	
	2.1	-1.03 ×
	GF+LB-R (Global file- and local block-level [3 replicas])	
	3.0	-2.87×
	GB-EC (Global block-level [Erasure coding])	
	6.9	-6.37×

Deduplication ratio vs. performance

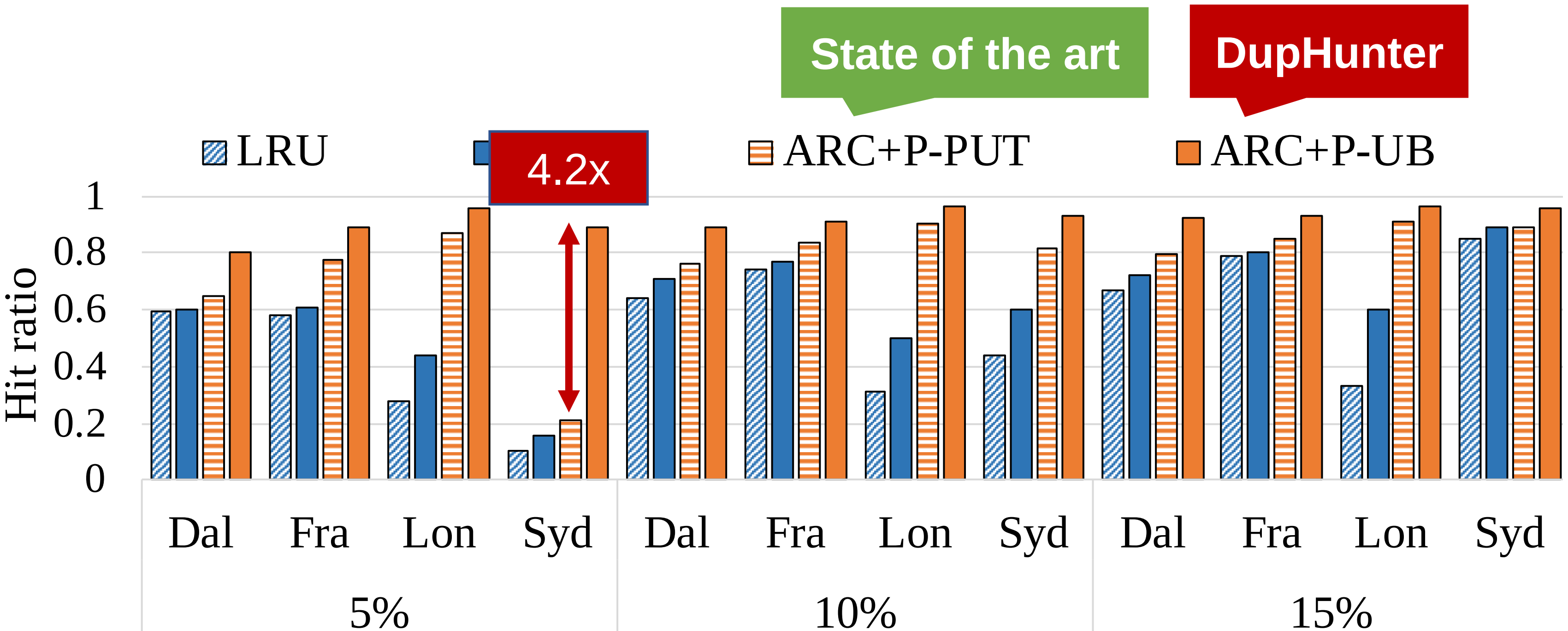
Mode	Dedup. ratio	Performance improvement (P-servers)
B-mode 1	1.5	1.6×
S-mode	1.3	2×
B-mode 2	1.2	2.6×
B-mode 3	1	2.8×

	Dedup ratio	Performance degradation (D-servers)
B-mode 0	**GF-R** (Global file-level [3 replicas])	
	2.1	-1.03 ×
	GF+LB-R (Global file- and local block-level [3 replicas])	
	3.0	-2.87×
	GB-EC (Global block-level [Erasure coding])	
	6.9	-6.37×

Prefetch cache hit ratio

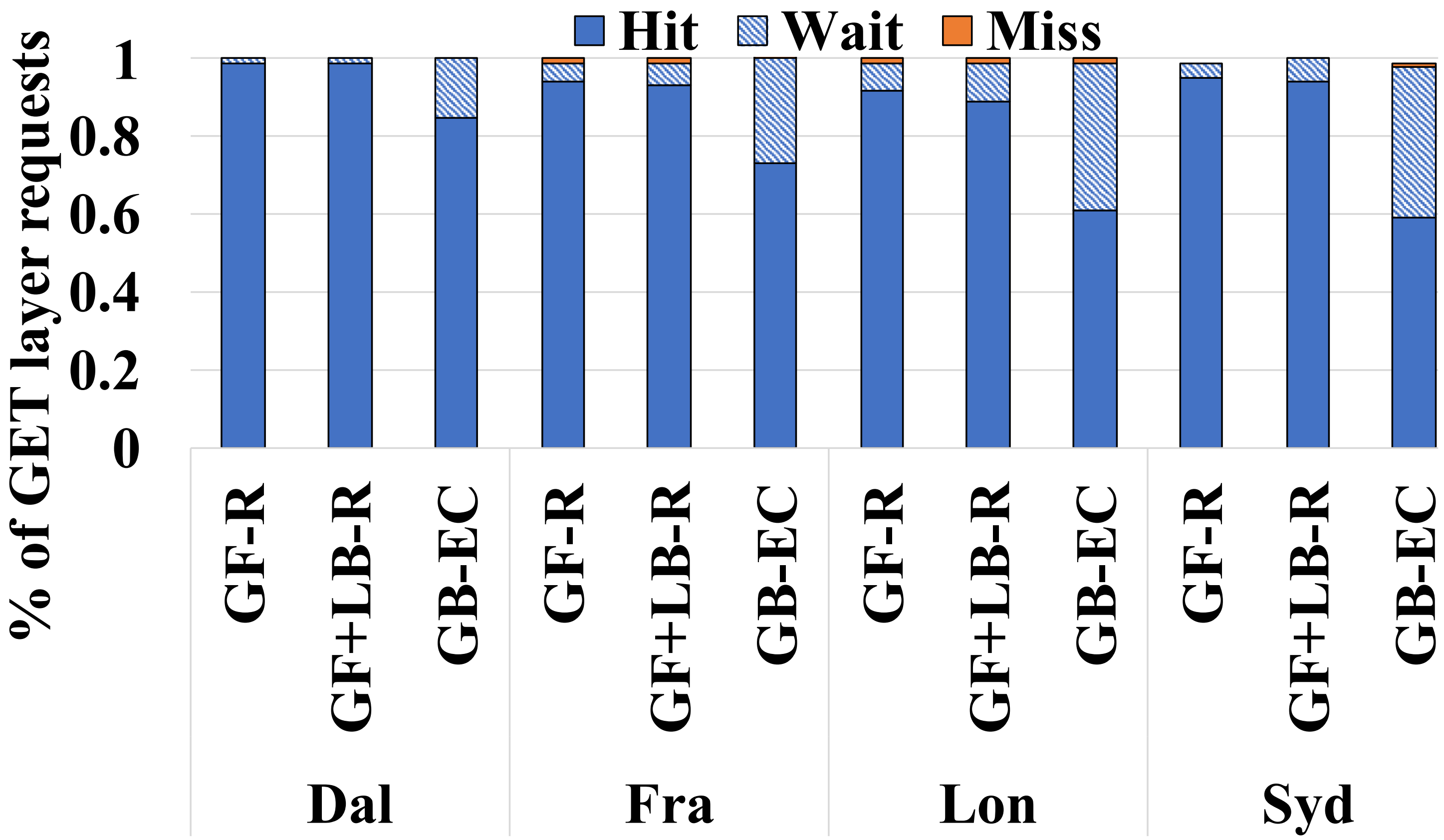


Prefetch cache hit ratio

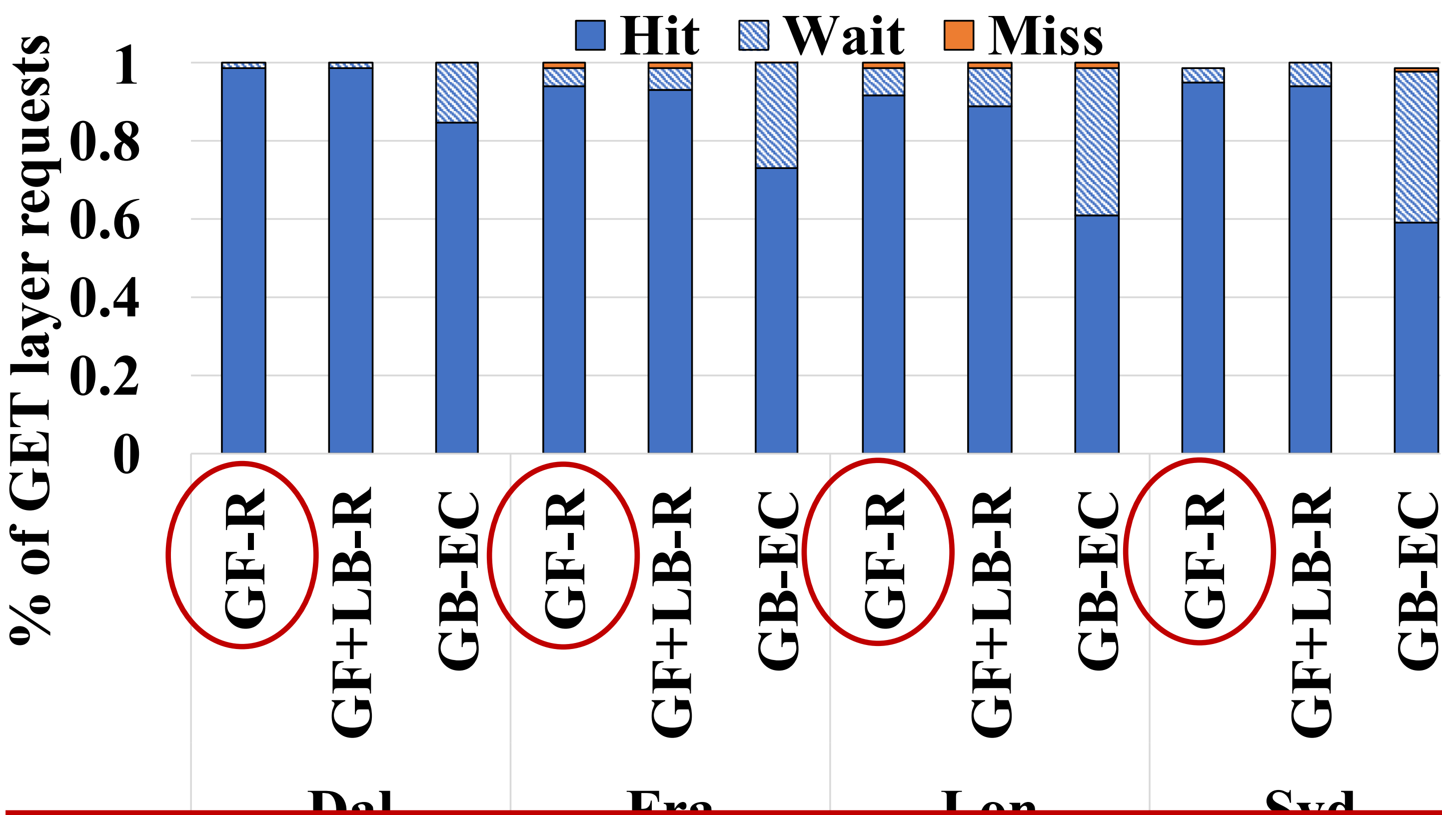


Duphunter can provide high hit ratio while reducing tail latency.

Preconstruct cache hit ratio



Preconstruct cache hit ratio



Global file level dedeuplication also has the lowest wait and miss ratios.

Summary

- DupHunter exploits the redundancy in container images along with predictable user access patterns to achieve high space savings with low layer restore overhead.
  - *It supports multiple replica deduplication modes.*
  - *It facilitates parallel layer reconstruction.*
  - *It offers proactive layer prefetching/preconstruction.*
- DupHunter reduces storage space needs by up to **6.9x** and can reduce the GET layer latency up to **2.8x** compared to the state of the art.
- DupHunter is available at *https://github.com/nnzhaocs/DupHunter*.

THANK YOU

Questions: Nannan Zhao znannan1@vt.edu

DSSL@VT: http://dssl.cs.vt.edu